Formal Specification and Integration of Distributed Security Policies

M. Mejri[1] and H. Yahyaoui[2]
[1]Computer Science Department, Laval University, Canada
mejri@ift.ulaval.ca
[2]Computer Science Department, Kuwait University, Kuwait
hamdi@cs.ku.edu.kw

Abstract

We propose in this paper the Security Policy Language (SePL), which is a formal language for capturing and integrating distributed security policies. The syntax of SePL includes several operators for the integration of policies and it is endowed with a denotational semantics that is a generic semantics, i.e., which is independent of any evaluation environment. We prove the completeness of SePL with respect to sets theory. Furthermore, we provide a formalization of a subset of the eXtensible Access Control Markup Language (XACML), which is the well-known standard informal specification language of Web security policies. We provide also a semantics for XACML policy combining algorithms.

Keywords: Security Policies; Formal Languages; Semantics; Integration; XACML.

1. Introduction

Nowadays, there is a drastic growing of security threats, which benefit from security breaches in systems to jeopardize their security and achieve malicious goals such as thief and illegal access to information, identity masquerading, etc. The consequences of security attacks can be fatal to institutions and companies, which made security a major concern for people in industry and academia. In this context, building secure systems is becoming a paramount challenge mainly in a distributed environment where each system has its own security policies, which may conflict with policies of other systems. In such environment, security policies specification is based on standard languages, which are often informal and complex such as the eXtensible Access Control Markup Language (XACML) [27]; the well-known standard informal specification language of Web security policies. Such complexity makes the learning curve of the proposed languages very high and increases the likelihood of having design errors. Accordingly, there is a desideratum for providing simple and formal models that capture such policies and allow to reason about them.

There are two main classes of approaches for formalizing security policies: Model and language based methods. Model based approaches leverage formalisms such as transition systems to capture policies. Model checking of system compliance to security properties is one of the

main targets behind the design of such models. The main issue with such methods is their limited scalability when applied to huge policies.

Regarding language based approaches, several languages were proposed to specify security policies. XML-based specification languages use XML tags to describe security policies and rules between subjects and resources. Famous XML-based specification languages include Security Assertion Markup Language (SAML) [20], XML Access Control Policy Specification Language (XACL) [8], and Extensible Access Markup Control Language (XACML) [27]. The issue with such languages is that they are machine readable and so difficult to be understood. Furthermore, they lack the formal aspect that allows reasoning about them. Declarative languages provide a high level of simplicity and readability for the specification of security policies. We mention Ponder [9] as a famous declarative, object-oriented language for specifying policies for the security and management of distributed systems. The main issue with such languages is the lack of abstractness that allows to reason about correctness and completeness issues. Event based languages, such as Policy Description Language (PDL) [7] and DEFCon Policy Language (DPL) [21], leverage events and actions to model security policies and rules between subjects and resources. Some of these languages put more focus on actions rather than data while others express constraints on event flows. The main issue with such languages is their complexity and sometimes their modeling of low level details. Algebraic languages allow to formally define security policies. An important feature of algebraic security policy languages is their simplicity, powerful expressiveness and compactness.

We advocate in this work the need for a simple, formal, precise and rigorous language to guarantee the absence of inconsistencies in policies, reason about their integration, and prove their correctness. To achieve this goal, we define in this paper a new language called Security Policy Language (SePL) for the specification of distributed security policies. We also show how the language can be leveraged to define the integration of policies. In addition, we present a formalization of a large subset of the latest version of XACML based on SePL, which provides a simple understanding of that language. The contributions of this paper are three-folds:

- The proposal of a new formal Security Policy Language (SePL) for the specification and integration of security policies.
- The elaboration of a generic denotational semantics for SePL that is capable of expressing complex security policies.
- The formalization of almost all XACML policy combining algorithms based on SePL and the proof of the completeness of SePL with respect to set theory.

The paper is organized as follows. Section 2 is dedicated to the presentation of the syntax and semantics of SePL. We provide a formalization of XACML based on SePL in Section 3. We provide in Section 5 a minimal version of SePL. Section 4 is devoted to the proof of completeness of SePL. In section 6, we provide a comparison of our work with the related work. Finally, we provide some concluding remarks in Section 7.

2. Security Policy Language: Syntax and Semantics

We present in this section the syntax and semantics of the Security Policy Language (SePL). We use some of the abbreviations and notations that we exhibit in the following. Let $\mathcal{A} =< a_1, \ldots, a_n >$ be a list of attributes such that the domain of a_i, $1 \leq i \leq n$, is denoted by $\mathcal{D}(a_i)$. Let $\varphi_1 = (d_1^1, \ldots, d_n^1)$ and $\varphi_2 = (d_1^2, \ldots, d_n^2)$ such that $d_i^1 \subseteq \mathcal{D}(a_i)$ and $d_i^2 \subseteq \mathcal{D}(a_i)$, for all i in $\{1, \ldots, n\}$. We assume also that a set d can be defined either by explicitly giving its elements (e.g $\{v_1 \ldots, v_m\}$) or by a predicate (e.g $\{x : \mathcal{D}(a) \mid x \neq v\}$). We assume that $\top$ denotes $(\mathcal{D}(a_1), \ldots, \mathcal{D}(a_n))$ and $\bot$ denotes $(\emptyset, \ldots, \emptyset)$. We assume also that the value of an attribute can stay unknown: for example $a_1 = x$ where x is variable in $\mathcal{D}(a_1)$. If the name of the variable is not important, we simply write $a_1 =?$. When there is no ambiguity, $?$ is also used to abbreviate $(?, \ldots, ?)$ which is a tuple where all its attributes have unknown values, i.e., $(x_1, \ldots, x_n)$.

2.1. Syntax

The syntax of SePL is given by the following BNF grammar:

$$
\begin{array}{rcl}
\mathsf{P}, \mathsf{P}_1, \mathsf{P}_2 & ::= & \varepsilon \mid 0 \mid 1 \mid \mathsf{R} \mid \neg P \mid \ulcorner P \mid \mathsf{P}_1 \cdot \mathsf{P}_2 \\
& \mid & \mathsf{P}_1 [\!] \mathsf{P}_2 \mid \mathsf{P}_1]\!] \mathsf{P}_2 \mid \mathsf{P}_1 \parallel \mathsf{P}_2 \\
& \mid & \mathsf{P}_1 + \mathsf{P}_2 \mid \mathsf{P}_1 - \mathsf{P}_2 \mid \mathsf{P}_1 \ominus \mathsf{P}_2 \\
\mathsf{R} & ::= & < \varphi_1, \varphi_2 >
\end{array}
$$

where

- ε denotes the empty policy.
- 0 denotes the policy that denies all actions.
- 1 denotes the policy that accepts all actions.
- $\mathsf{R} =< \varphi_1, \varphi_2 >$ denotes the policy that accepts actions accepted by φ_1 and not denied by φ_2 and denies actions denied by φ_2 and not accepted by φ_1.
- $\neg P$ denotes the policy that accepts actions that P denies and denies actions that P accepts.
- $\ulcorner P$ behaves like P except that it transforms the indeterminate part of P to not applicable. In other words, if the accept part or the deny part of P is indeterminate, it becomes an empty set.
- $\mathsf{P}_1 \cdot \mathsf{P}_2$ denotes the sequential composition of policies. This gives the result of the first one applicable. If no one is applicable, the result is either not applicable or indeterminate if P_1 and P_2 are indeterminate.
- $\mathsf{P}_1 [\!] \mathsf{P}_2$ denotes the policy that gives priority to accept, i.e, it accepts an action when P_1 or P_2 accepts it and denies an action when no one of them accepts it and at least one of them denies it. Otherwise, the policy is either not applicable or indeterminate if P_1 or P_2 is indeterminate.

- $\mathsf{P}_1 \Vdash \mathsf{P}_2$: This is the dual part of $\Vdash$ since it gives priority to deny. It denies an action when P_1 or P_2 denies it and accepts an action when no one of them denies it and at least one of them accepts it. Otherwise, the policy is either not applicable or indeterminate if P_1 or P_2 is indeterminate.

- $\mathsf{P}_1 \parallel \mathsf{P}_2$ denotes the parallel composition of policies. It accepts an action when both of them accept and denies when both of them deny. Otherwise, the policy is either not applicable or indeterminate if P_1 or P_2 is indeterminate.

- $\mathsf{P}_1 + \mathsf{P}_2$ denotes the choice between two policies. It accepts when one of them accept and no one denies, and denies when one of them denies and no one accepts. Otherwise, the policy is either not applicable or indeterminate if P_1 or P_2 is indeterminate.

- $\mathsf{P}_1 - \mathsf{P}_2$ denotes the policy that accepts when P_1 accepts and P_2 is not applicable and denies when P_1 denies and P_2 is not applicable. Otherwise, the policy is either not applicable or indeterminate if P_1 or P_2 is indeterminate.

- $\mathsf{P}_1 \ominus \mathsf{P}_2$. This behaves like $\mathsf{P}_1 - \mathsf{P}_2$ except that the result is indeterminate ("?") when there is an overlap between P_1 and P_2 (i.e. the accepts of P_1 together with its denies is not disjoint from the accepts of P_2 added to its denies) .

To reduce the number of parentheses when writing properties, we assume the following precedence between operators (from strong to weak) $\neg$, $\ulcorner$, ., $||$, $\Vdash$, $\Vdash$, $+$. For example $P_1 + \neg P_2.P_3||P_4$ is same as $P_1 + (((\neg P_2).P_3)||P_4)$. Notice that SePL does not explicitly contain the composition rules (combining algorithms) used in XACML, but, as we show later that, it is expressive enough to specify them. Also, the operators of SePL are not independent from each others. We show later that we can remove many of them without affecting the expressiveness of the language. In fact, we can keep only the operators belonging to the set $\{\neg, \ulcorner, \|, +, \ominus\}$ without affecting the expressiveness of the language.

2.2. Semantics

- **Preliminary definitions:** Let $A = (\alpha_1, \ldots, \alpha_n)$ and $B = (\beta_1, \ldots, \beta_n)$ be two tuples of n set elements. In the sequel, we define the following semantic operators:

$$\begin{array}{lcl} A \cup B & = & (\alpha_1 \cup \beta_1, \ldots, \alpha_n \cup \beta_n) \\ A \cap B & = & (\alpha_1 \cap \beta_1, \ldots, \alpha_n \cap \beta_n) \\ A - B & = & (\alpha_1 - \beta_1, \ldots, \alpha_n - \beta_n) \\ A \ominus B & = & (\alpha_1 \ominus \beta_1, \ldots, \alpha_n \ominus \beta_n) \\ A\urcorner & = & (\alpha_1\urcorner, \ldots, \alpha_n\urcorner) \\ \ulcorner A & = & (\ulcorner\alpha_1, \ldots, \ulcorner\alpha_n) \end{array}$$

where

$$\begin{aligned}\alpha \ominus \beta &= \alpha - \underline{((\alpha \cap \beta) \cap ?)} \\ &= \alpha \cap \overline{((\alpha \cap \beta) \cap ?)} \\ &= \begin{cases} \alpha & \text{if } \alpha \cap \beta = \emptyset \\ ? & \text{otherwise} \end{cases}\end{aligned}$$

$$\ulcorner\alpha = \begin{cases} \alpha & \text{if } \alpha \neq ? \\ \emptyset & otherwise \end{cases}$$

$$\alpha\urcorner = \begin{cases} \alpha & \text{if } \alpha \neq ? \\ \mathcal{D}(\alpha) & otherwise \end{cases}$$

$$\begin{aligned}\ulcorner(\alpha \cup \beta) &= \ulcorner\alpha \cup \ulcorner\beta \\ \ulcorner(\alpha \cap \beta) &= \ulcorner\alpha \cap \ulcorner\beta \\ \ulcorner(\overline{\alpha}) &= \overline{\alpha\urcorner} \\ (\alpha \cup \beta)\urcorner &= \alpha\urcorner \cup \beta\urcorner \\ (\alpha \cap \beta)\urcorner &= \alpha\urcorner \cap \beta\urcorner \\ (\overline{\alpha})\urcorner &= \overline{\ulcorner\alpha}\end{aligned}$$

- **Absolute Semantics ($[\![\ - \ ]\!]$):** The absolute semantics of a policy P, denoted by $[\![P]\!]$ returns a pair (A, B) where A is the acceptance domain of P and B is its denying domain. The domain that is not explicitly accepted or denied by a policy defines implicitly its "non-applicable" domain. More formally, $[\![-]\!]$ is inductively defined as shown by Table 1 where (A_1, D_1) denotes the semantics of P_1, (A_2, D_2) denotes the semantics of P_2 and (A, D) denotes the semantics of P.

  The absolute semantics gives the meaning of a policy independent from the environment in which it will be evaluated. One of the advantages of this semantics is that any optimization or simplification applied to it can be used for any environment. This means that we can reduce the time of the evaluation of the semantics by optimizing this semantics one time and use it many times with different environments. Another advantage is that it allows to prove some general results independent from a specific environment.

- **Relative Semantics ($[\![\ - \ ]\!]_\Gamma$):** The semantics of a policy P relative to an environment Γ, denoted by $[\![P]\!]_\Gamma$, returns the decision of the policy P regarding Γ. In practice, the environment Γ contains both the action that we want to execute together with its context (the values of the variables of the environment during its execution). For our semantics, the result is (α, β) where $\{\alpha, \beta\} \subset \{\mathsf{T}, ?, \mathsf{F}\}$ (T stands for True, F stands for False, and ? stands for unknown). Usually, the result is considered as permit if $\alpha = \mathsf{T}$, deny if $\beta = \mathsf{T}$, otherwise the result is either not applicable or indeterminate. It is non applicable if $\alpha = \beta = \mathsf{F}$ and it is indeterminate if α and β are in $\{(\mathsf{F}, ?), (?, \mathsf{F}), (?, ?)\}$. More details about these notations will be given within the section regarding the formalization of the XACML language.

Table 1: Absolute Semantics

$$
\begin{aligned}
[\![< \varphi_1, \varphi_2 >]\!] &= (\varphi_1 - \varphi_2, \varphi_2 - \varphi_1) \\
[\![\varepsilon]\!] &= (\bot, \bot) \\
[\![0]\!] &= (\bot, \top) \\
[\![1]\!] &= (\top, \bot) \\
[\![\neg \mathsf{P}]\!] &= (D, A) \\
[\![\lceil \mathsf{P}]\!] &= (\lceil A, \lceil D) \\
[\![\mathsf{P_1} \cdot \mathsf{P_2}]\!] &= (A_1 \cup (A_2 - D_1), D_1 \cup (D_2 - A_1)) \\
[\![\mathsf{P_1} \lfloor\!\lfloor \mathsf{P_2}]\!] &= (A_1 \cup A_2, (D_1 - A_2) \cup (D_2 - A_1)) \\
[\![\mathsf{P_1} \rfloor\!\rfloor \mathsf{P_2}]\!] &= (A_1 - D_2 \cup A_2 - D_1, D_1 \cup D_2) \\
[\![\mathsf{P_1} \parallel \mathsf{P_2}]\!] &= (A_1 \cap A_2, D_1 \cap D_2) \\
[\![\mathsf{P_1} + \mathsf{P_2}]\!] &= ((A_1 \cup A_2) - (D_1 \cup D_2), (D_1 \cup D_2) - (A_1 \cup A_2)) \\
[\![\mathsf{P_1} - \mathsf{P_2}]\!] &= (A_1 - (A_2 \cup D_2), D_1 - (A_2 \cup D_2)) \\
[\![\mathsf{P_1} \ominus \mathsf{P_2}]\!] &= (A_1 \ominus (A_2 \cup D_2), D_1 \ominus (A_2 \cup D_2))
\end{aligned}
$$

More formally, let $\Gamma = [a_1 = v_1, \ldots, a_n = v_n]$ be an environment where $v_i \in \mathcal{D}(a_i) \cup \{?\}$ and "$a =?$" means that the value of the attribute a is unknown. Let P be a policy such that $[\![P]\!] = (A, D)$. We extend the definition of $[\![-]\!]$, as follows:

$$[\![P]\!]_\Gamma = ([\![A]\!]_\Gamma, [\![D]\!]_\Gamma)$$

$$[\![(d_1, \ldots, d_n)]\!]_\Gamma = [\![d_1]\!]_{\Gamma(a_1)} \wedge \ldots \wedge [\![d_n]\!]_{\Gamma(a_n)}$$

$$[\![d]\!]_{\Gamma(a)} = \begin{cases} \mathsf{T} & \text{if } \Gamma(a) \in d \text{ or } d = \mathcal{D}(a) \\ \mathsf{F} & \text{if } \Gamma(a) \notin d \text{ or } d = \emptyset \\ ? & \text{otherwise} \end{cases}$$

Notice that the truth tables of the three-valued logic is as shown hereafter:

$\wedge$	T	?	F
T	T	?	F
?	?	?	F
F	F	F	F

$\vee$	T	?	F
T	T	T	T
?	T	?	?
F	T	?	F

$\ominus$	T	?	F
T	?	?	T
?	?	?	?
F	F	F	F

b	$\neg b$
T	F
?	?
F	T

b	$\bar{b}$
T	T
?	F
F	F

It is sometimes useful to define $[\![-]\!]_\Gamma$ in a compositional way as shown in Table 2 where (a_1, d_1) denotes $[\![P_1]\!]_\Gamma$, (a_2, d_2) denotes $[\![P_1]\!]_\Gamma$ and (a, d) denotes $[\![P]\!]_\Gamma$:

where $a - b$ denotes $a \wedge \neg b$

2.3. Three-Valued Logic: choices and consequences

The first famous three-valued logic has been proposed by Lukasiewicz in [1] since 1920. A formula can be evaluated to true (T), false (F) or undecided (?). This third value makes the logic complicated and we need to make some choices about the evaluation of the combination of formula using classical boolean operator. For example, the result of "$? \vee \neg ?$" is it true? the result of "$? \wedge \neg ?$" is it true? is $x \rightarrow y$ is equal to $x \vee \neg y$?. Beside the logic proposed by Lukasiewicz, many others were proposed to reflect these different choices. The Kleene three-valued logic [2] and the Fitting logic [5] are the most famous. Within the Lukasiewicz, Kleene and Fitting logics, we have the following properties:

- **Idempotency** ($x \vee x \equiv x$ and $x \wedge x \equiv x$);

Table 2: Relative Semantics

$$
\begin{array}{rcl}
[\![< \varphi_1, \varphi_2 >]\!]_\Gamma & = & ([\![\varphi_1 - \varphi_2]\!]_\Gamma, [\![\varphi_2 - \varphi_1]\!]_\Gamma) \\
[\![\varepsilon]\!] & = & (\mathsf{F}, \mathsf{F}) \\
[\![0]\!]_\Gamma & = & (\mathsf{F}, \mathsf{T}) \\
[\![1]\!]_\Gamma & = & (\mathsf{T}, \mathsf{F}) \\
[\![\neg \mathsf{P}]\!]_\Gamma & = & (d, a) \\
[\![\ulcorner \mathsf{P}]\!]_\Gamma & = & (\ulcorner a, \ulcorner d) \\
[\![\mathsf{P_1} \cdot \mathsf{P_2}]\!]_\Gamma & = & (a_1 \vee (a_2 - d_1), d_1 \vee (d_2 - a_1)) \\
[\![\mathsf{P_1} \lfloor\!\lfloor \mathsf{P_2}]\!]_\Gamma & = & (a_1 \vee a_2, (d_1 - a_2) \vee (d_2 - a_1)) \\
[\![\mathsf{P_1} \rfloor\!\rfloor \mathsf{P_2}]\!]_\Gamma & = & (a_1 - d_2 \vee a_2 - d_1, d_1 \vee d_2) \\
[\![\mathsf{P_1} \parallel \mathsf{P_2}]\!]_\Gamma & = & (a_1 \wedge a_2, d_1 \wedge d_2) \\
[\![\mathsf{P_1} + \mathsf{P_2}]\!]_\Gamma & = & ((a_1 \vee a_2) - (d_1 \vee d_2), (d_1 \vee d_2) - (a_1 \vee a_2)) \\
[\![\mathsf{P_1} - \mathsf{P_2}]\!]_\Gamma & = & (a_1 - (a_2 \vee d_2), d_1 - (a_2 \vee d_2)) \\
[\![\mathsf{P_1} \ominus \mathsf{P_2}]\!]_\Gamma & = & (a_1 \ominus (a_2 \vee d_2), d_1 \ominus (a_2 \vee d_2))
\end{array}
$$

- **Commutativity** ($x \vee y \equiv y \vee x$ and $x \wedge y \equiv y \wedge x$);
- **Associativity** ($(x \vee y) \vee z \equiv x \vee (y \vee z)$ and $(x \wedge y) \wedge z \equiv x \wedge (y \wedge z)$);
- **Absorption** ($(x \wedge y) \vee x \equiv x$ and $(x \vee y) \wedge x \equiv x$);
- **Distributivity** ($(x \wedge y) \vee z \equiv (x \wedge z) \vee (y \wedge z)$ and $(x \vee y) \wedge z \equiv (x \vee z) \wedge (y \vee z)$);
- **Double Negation** ($\neg\neg x \equiv x$);
- **De Morgan** ($\neg(x \vee y) \equiv \neg x \wedge \neg y$ and $\neg(x \wedge y) \equiv \neg x \vee \neg y$);
- **Contraposition** ($x \rightarrow y \equiv \neg y \rightarrow \neg x$).

Lukasiewicz and Kleene logic have the **Equivalence** ($x \leftrightarrow y \equiv (x \rightarrow y) \wedge (y \rightarrow \neg x)$) but not the Fitting one. The Kleene and Fitting logic have the **Syllogism** ($(x \leftrightarrow y) \wedge (y \rightarrow z) \equiv (x \rightarrow z)$) but not the Lukasiewicz one. None of the the previous logics has the **Excluded Middle** ($(x \vee \neg x) \equiv T$) or the **Contradiction** ($(x \wedge \neg x) \equiv F$). For our semantics, we adopt the following convention: The value "?" is used to represent a formula that can neither be evaluated to true (T) nor to false (F) due to some missing values for a part of its variables. For example, $x \wedge y$ can be considered as ? if the values of x and y are unknown. However, $x \wedge (\neg x \wedge y)$ will be evaluated to F even x and y are unknown since $x \wedge (\neg x \wedge y) \equiv (x \wedge \neg x) \wedge y) \equiv F \wedge x \equiv F$. Notice that the problem of knowing whether a formula is valid or not is decidable. A symbol "?" can be substituted by a fresh variable in a formula.

3. Formalizing XACML Policies based on SePL

We show in this section how we can leverage SePL to formalize XACML policies.

3.1. Overview of XACML

XACML (eXtensible Access Control Markup Language) [27] is a set of XML schemas that define the specifications of a language for access control policies. As shown in Figure 3.1, a XACML policy is composed of a target, an obligation and a set of rules. Rules are also composed of target conditions and effects or permissions.

Because a policy may contain multiple rules with different decisions, we need to clarify how to build the decision of a policy from the decision of its rules. To this end, we formalized the "Rules Combining Algorithm" that was proposed for this purpose by OASIS. It is also possible to aggregate policies to form a "PolicySet". A *Policyset* has also a target that limits the scope of its applicability and an algorithm that defines its global decision from the local decisions returned by its policies. The target of an access request is first compared to the target of a *Policyset*, if they are different this *Policyset* is not applicable. Otherwise, the target of the request is compared to the targets of policies inside the *Policyset*. A policy is qualified as not applicable when its target is different from the target of the request. When a policy and a request agree on the target, the request

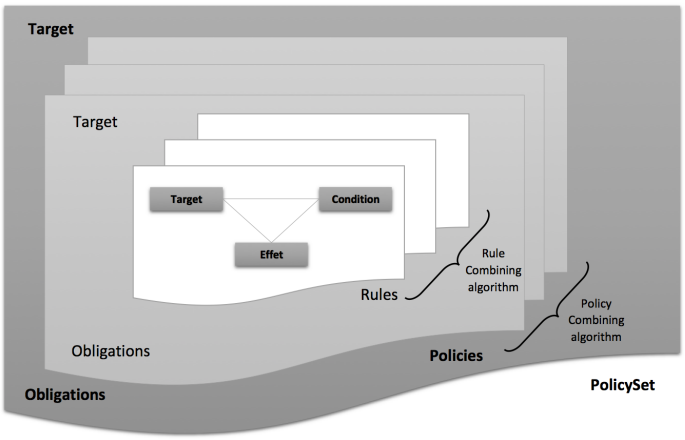

Figure 3.1: XACML structure

is analyzed by the rules inside the policy: A rule is applicable if its targets include the target of the request and its condition is evaluated to true.

The XACML standard describes four types of Combining Algorithms. Hereafter, we describe how a *PolicySet* combines the results of its policies. The same algorithms can be used to combine the decisions of rules to build the decision of their policies.

- **Permit-overrides**: A PolicySet accepts if at least one of its policies accepts and denies if no one of its policies accepts and at least one denies. Otherwise, the PolicySet is not applicable.
- **Deny-overrides**: It returns "deny" if at least one policy denies and it returns "accept" if no policy denies and at least one policy accepts. Otherwise, the policy is not applicable.
- **First Applicable**: It accepts if there is at least one policy that accepts and this policy is not proceeded by a denying one and vise-versa.
- **Only-one-applicable**: if more than one policy is applicable, then the result is indeterminate. Otherwise, the result of the unique applicable policy will be considered.

3.2. Abbreviations

For the sake of making the presentation clear, we adopt the following abbreviations:

- We use the terms N/A, Permit, Deny, Indeterminate(P), Indeterminate(D), Indeterminate(PD) as an abbreviation of the following situations:

$$\begin{aligned} \mathsf{N/A} &= (\mathsf{F},\mathsf{F}) \\ \mathsf{Permit} &= (\mathsf{T},\mathsf{F}) \text{ or } (\mathsf{T},?) \\ \mathsf{Deny} &= (\mathsf{F},\mathsf{T}) \text{ or } (?,\mathsf{T}) \\ \mathsf{Indeterminate(P)} &= (?,\mathsf{F}) \\ \mathsf{Indeterminate(D)} &= (\mathsf{F},?) \\ \mathsf{Indeterminate(PD)} &= (?,?) \end{aligned}$$

- We denote by $\varphi \rightarrow \mathsf{p}$ and $\varphi \rightarrow \mathsf{d}$ the following policies:

$$\begin{aligned} \varphi \rightarrow \mathsf{p} &\equiv (\varphi, \bot) \\ \varphi \rightarrow \mathsf{d} &\equiv (\bot, \varphi) \end{aligned}$$

- Any $\varphi = (d_1, \ldots, d_n)$ can be represented by its restricted attributes only. An attribute a_i is restricted in φ if $d_i \neq \mathcal{D}(a_i)$. For example, if $\mathcal{A} =< \mathsf{Role}, \mathsf{Object}, \mathsf{Action} >$ such that $\mathcal{D}(\mathsf{Role}) = \{r_1, r_2, r_3\}$, $\mathcal{D}(\mathsf{Object}) = \{o_1, o_2, o_3, o_4\}$ and $\mathcal{D}(\mathsf{Action}) = \{a_1, a_2\}$, then we can use the following abbreviation:

  (1)

$$\begin{aligned} & (\{r_1, r_2, r_3\}, \{o_1\}, \{a_2\}) \\ \equiv \\ & (\mathsf{Object} \in \{o_1\}, \mathsf{Action} \in \{a_2\}) \\ \equiv \\ & (\mathsf{Object} = o_1, \mathsf{Action} = a_2) \end{aligned}$$

  (2)

$$\begin{aligned} & (\{r_1, r_2, r_3\}, \{o_1\}, \{a_1, a_2\}) \\ \equiv \\ & (\mathsf{Object} \in \{o_1\}) \\ \equiv \\ & (\mathsf{Object} = o_1) \end{aligned}$$

- If op is a binary SePL operator, we denote by $op(P_1, P_2)$ the prefix notation of $P_1 \ op \ P_2$. For example, $P_1 \| P_2$ can be denoted by $\|(P_1, P_2)$. The result can be generalized to n compositions since all the operators of SePL are transitive. For example, $P_1 \| P_2 \| \ldots \| P_n$ can be represented by $\|(P_1, P_2, \ldots, P_n)$.

- The policy $\phi : P$ is the abbreviation defined inductively as follows:

$$
\begin{array}{lcl}
\phi : (\phi_1, \phi_2) & = & (\phi \cap \phi_1, \phi \cap \phi_2) \\
\phi : \neg P & = & \neg(\overline{\phi} : P) \\
\phi :^{\ulcorner} P & = & ^{\ulcorner}(\phi : P) \\
\phi : (\mathsf{P}_1 \parallel \mathsf{P}_2) & = & (\phi : \mathsf{P}_1) \parallel (\phi : \mathsf{P}_2) \\
\phi : (\mathsf{P}_1 + \mathsf{P}_2) & = & (\phi : \mathsf{P}_1) + (\phi : \mathsf{P}_2) \\
\phi : (\mathsf{P}_1 \ominus \mathsf{P}_2) & = & (\phi : \mathsf{P}_1) \ominus (\phi : \mathsf{P}_2)
\end{array}
$$

Also, we update the precedence between operators (from strong to weak) ":", "$\neg$", "$\ulcorner$", ".", "$||$", "$[\!|$", "$|\!]$", "$+$".

3.3. From XACML to SePL

Table 3.3 outlines a BNF grammar that we propose to capture a significant subset of XACML-3.0. The literature does not contain such grammar and it is not simple to build it from XACML-3.0 specification. This will be useful also to automatically build a lexical analyzer and parser for XACML-3.0 using tools such as Lex and Yacc [6].

The transformation function $\lceil - \rceil$ from XACML to SePL can be inductively defined as shown in Table 3.4. Its definition is based on our own understanding of the XACML-3.0 textual description. Each rule is dedicated to a specific component (PolicySet, Policy, Rules, etc.) of XACML-3.0, where bold terms denote terminal words. For example, transforming an XACML condition (described by "$<$ **Condition** $>$ $BooleanExpression$ $<$ / **Condition** $>$") to SePL turns to extract the $BooleanExpression$ and ignore the rest.

We have also developed, using PHP and XML, a web based application allowing to automatically convert XACML security policies to SPL as shown by Figure 3.2.

Figure 3.2: Automatic Conversion of XACML to SPL

3.4. Example

The XACML policy, given in Table 4, includes two rules and is extracted from [27]: the first one provides the reading right of the file secret.txt to only Alice or Bob while the second rule provides access for anyone to that file. The SePL term related to this XACML policy is as follows:

Table 3: A BNF Grammar for a Subset of XACML-3.0

PDPpolicies ::= *PolicySet* | *Policy*

PolicySet ::= < **POLICYSET** *Pheader* > [*Description*] *Targets Policies* [*Obligation*] [*Advice*] < / **POLICYSET** >
Policy ::= < **POLICY** *Rheader* > [*Description*] *Targets Rules* [*Obligation*] [*Advice*] < / **POLICY** >
Policies ::= *Policy* | *Policy Policies*
Rules ::= < **RULE** *Rheader* > [*Description*] [*Targets*] [*Condition*] [*Obligation*] [*Advice*] < / **RULE** >

PSheader ::= **PolicySetId** = *string* **Version** = *number* **PolicyCombiningAlgId** = *Palg*
Pheader ::= **PolicyId** = *string* **Version** = *number* **RuleCombiningAlgId** = *Ralg*
Rheader ::= **RuleId** = *string* **Effect** = *REffect*

Palg ::= **only-one-applicable**
| *Ralg*

Ralg ::= **deny-overrides**
| **permit-overrides**
| **first-applicable**
| **ordered-permit-overrides**

REffect ::= **Permit** | **Deny**

Targets ::= < **TARGET** > [*MatchAny*] < / **TARGET** >

MatchAny ::= < **AnyOf** > *matchAll* < / **AnyOf** >
|< **AnyOf** > *matchAll* < / **AnyOf** > *MatchAny*

MatchAll ::= < **AllOf** > *Matches* < / **AnyOf** >
|< **AnyOf** > *Matches* < / **AllOf** > *MatchAll*

Matches ::= *Match* | *Match Matches*

Match ::= < **Match MatchID =** *MatchId* >
< **AttrValue** > *value* < / **AttrValue** >
< **AttributeDesignator** *ADHeader* / >
< / **Match** >

MatchId ::= **string-equal**
| **integer-equal**
| **string-regexp-match**
| **integer-greater-than**
| ...

ADHeader ::= **Category=** *Subject* **AttributeId =** *AttSubject* **DataType=** *type* **MustBePresent=** *boolean*
| **Category= resource AttributeId =** *AttResource* **DataType=** *type* **MustBePresent=** *boolean*
| **Category= action AttributeId =** *AttAction* **DataType=** *type* **MustBePresent=** *boolean*
| **Category= environment AttributeId =** *AttEnv* **DataType=** *type* **MustBePresent=** *boolean*

Subject ::= **access-subject**
| **recipient-subject**
| **intermediary-subject**
...

AttSubject ::= **subject-id**
| **subject-id-qualifier**
| **key-info**
| **authentication-time**
...

AttResource ::= **resource-id**
| **target-namespace**

AttAction ::= **action-id**
| **implied-action**
| **action-namespace**

AttEnv ::= **current-time**
| **current-date**
| **current-dateTime**

type ::= **x500Name**
| **rfc822Name**
| **ipAddress**
| **dnsName**
| **xpathExpression**
| **string**
| **boolean**
| **double**
| **time**
| **date**
| **dateTime**
| **anyURI**
| **hexBinary**
| **base64Binary**

Condition ::= < **Condition** > *BooleanExpression* < / **Condition** >

$P = \phi : (P_1.P_2)$
$\phi = string - equal(resource - id, secret.txt)$
$P_1 = (string - equal(action - id, write) \rightarrow \mathbf{d}$
$= (\perp, (string - equal(action - id, write))$
$P_2 = (string - equal(subject - id, Alice) \rightarrow \mathbf{d}$
$= (\perp, (string - equal(subject - id, Alice))$

3.5. SePL formal semantics for XACML combining algorithms

We provide in this section a SePL-based semantics for XACML combining algorithms. The proofs of the conformance of our semantics for each XACML combining algorithm and its XACML specification are provided in the appendix.

3.5.1. Permit-override:

Permit-overrides between $P_1, \ldots, P_n$, denoted by $POR(P_1, \ldots, P_n)$: It accepts if at least one policy accepts and denies if no one accept and at least one denies. It can be formalized in SePL as follows:

$$POR(P_1, \ldots, P_n) \approx P_1 \Vert \ldots \Vert P_n$$

3.5.2. Deny-overrides:

Deny-overrides between $P_1, \ldots, P_n$, denoted by $DOR(P_1, \ldots, P_n)$: It denies if at least one policy denies and accepts if no one denies and at least one accepts. It can be formalized in SePL as follows:

$$DOR(P_1, \ldots, P_n) \approx P_1 \| \ldots \| P_n$$

3.5.3. First-Applicable:

First-Applicable between $P_1, \ldots, P_n$, denoted by $FA(P_1, \ldots, P_n)$: It accepts if there is at least one policy that accepts and that is not proceeded by a denying one and vise-versa. It can be formalized in SePL as follows:

$$FA(P_1, \ldots, P_n) \approx P_1 \ldots P_n$$

.

3.5.4. Only-one-applicable:

Only-one-applicable between $P_1, \ldots, P_n$, denoted by $OOA(P_1, \ldots, P_n)$: if more than one policy is applicable, the result will be neither accept nor deny (without decision). Otherwise, the unique applicable policy will be applied. It is formalized in SePL as follows:

$OOA(P_1, \ldots, P_n) \approx (P_1 \ominus \Sigma_{i=2}^{n} P_i) + \ldots + (P_j \ominus \Sigma_{i=1, i \neq j}^{n} P_i) + \ldots + (P_n \ominus \Sigma_{i=1}^{n-1} P_i)$

```
<Policy  PolicyId="SimplePolicy1"
        Version="1.0"  RuleCombiningAlgId="first-applicable">
        <Description>Access control policy for "secret.txt"  file</Description>
           <Target>
                 <AnyOf>
                      <AllOf>
                          <Match MatchId="string-equal">
                          <AttributeValue >secret.txt</AttributeValue>
                           <AttributeDesignator MustBePresent="false"
                                        Category="resource"
                                        AttributeId="resource-id"
                                        DataType="string"/>
                                   </Match>
                             </AllOf>
                    </AnyOf>
         </Target>
         <Rule RuleId= "SimpleRule1" Effect="Deny">
                      <Description> Don't allow write in secret.txt
                       </Description>
                       <Target>
                           <AnyOf>
                                <AllOf>
                                    <Match MatchId="string-equal">
                                    <AttributeValue >write</AttributeValue>
                                    <AttributeDesignator MustBePresent="false"
                                                Category="action"
                                                AttributeId="action-id"
                                                DataType="string"/>
                                      </Match>
                                 </AllOf>
                             </AnyOf>
                      </Target>
         </Rule>
         <Rule RuleId= "SimpleRule2" Effect="Deny">
                      <Description> Alice cannot read "secret.txt"
                       </Description>
                       <Target>
                           <AnyOf>
                                <AllOf>
                                    <Match MatchId="string-equal">
                                    <AttributeValue >Alice</AttributeValue>
                                    <AttributeDesignator MustBePresent="false"
                                                Category="access-subject"
                                                AttributeId="subject-id"
                                                DataType="string"/>
                                      </Match>
                                 </AllOf>
                             </AnyOf>
                             <AnyOf>
                                <AllOf>
                                    <Match MatchId="string-equal">
                                    <AttributeValue >read</AttributeValue>
                                    <AttributeDesignator MustBePresent="false"
                                                Category="action"
                                                AttributeId="action-id"
                                                DataType="string"/>
                                      </Match>
                                 </AllOf>
                             </AnyOf>
                      </Target>
         </Rule>
    </Policy>
```

Table 4: Example of XACML policies

Table 5: From XACML-3.0 to SePL

	$\lceil - \rceil : \ XACML-3.0 \ \longrightarrow \ PSL$
$(PolicySet)$	$\lceil <$ **POLICYSET** $Pheader >$ $[Description]$ $Targets$ $Policies$ $[Obligation]$ $[Advice]$ $</$ **POLICYSET** $> \rceil$ $= \lceil Targets \rceil$ **:** $\lceil Pheader \rceil$**(** $\lceil Policies \rceil$**)**
$(Policy)$	$\lceil <$ **POLICY** $Rheader >$ $[Description]$ $Targets$ $Rules$ $[Obligation]$ $[Advice]$ $</$ **POLICY** $> \rceil$ $= \lceil Targets \rceil$ **:** $\lceil Rheader \rceil$**(**$\lceil Rules \rceil$**)**
$(Policies)$	$\lceil Policy \ \ Policies \rceil = \lceil Policy \rceil, \lceil Policies \rceil$
$(Rules)$	$\lceil <$ **RULE** $Rheader >$ $[Description]$ $[Targets]$ $[Condition]$ $[Obligation]$ $[Advice] </$ **RULE** $> \rceil$ $=$ **(**$[\lceil Targets \rceil]$ $\lceil [Condition] \rceil$**)**$\lceil Rheader \rceil$
$PSheader$	$\lceil$ **PolicySetId** $=$ $string$ **Version** $=$ $number$ **PolicyCombiningAlgId** $=$ $Palg \rceil = \lceil Palg \rceil$
$(Pheader)$	$\lceil$ **PolicyId** $=$ $string$ **Version** $=$ $number$ **RuleCombiningAlgId** $=$ $Ralg \rceil = \lceil Ralg \rceil$
$(Rheader)$	$\lceil$ **RuleId** $=$ $string$ **Effect** $=$ $REffect \rceil = \rightarrow \lceil REffect \rceil$
$(Palg)$	$\lceil$ **only-one-applicable** $\rceil =$ **OOA** $\lceil$ **deny-overrides** $\rceil =$ **DO** $\lceil$ **permit-overrides** $\rceil =$ **PO** $\lceil$ **first-applicable** $\rceil =$ **FA** $\lceil$ **ordered-permit-overrides** $\rceil =$ **OPO**
$(REffect)$	$\lceil$ **Permit** $\rceil =$ **p** $\lceil$ **Deny** $\rceil =$ **d**
$(Targets)$	$\lceil <$ **TARGET** $> \ [MatchAny] </$ **TARGET** $> \rceil = \lceil [MatchAny] \rceil$
$(MatchAny)$	$\lceil <$ **AnyOf** $> matchAll </$ **AnyOf** $> \rceil = \lceil matchAll \rceil$ $\lceil <$ **AnyOf** $> matchAll </$ **AnyOf** $> MatchAny \rceil = \lceil matchAll \rceil \lceil MatchAny \rceil$
$(MatchAll)$	$\lceil <$ **AllOf** $> Matches </$ **AnyOf** $> \rceil = \lceil Matches \rceil$ $\lceil <$ **AnyOf** $> Matches </$ **AllOf** $> MatchAll \rceil = \lceil Matches \rceil \lceil MatchAll \rceil$
$(Matches)$	$\lceil Match \ \ Matches \rceil = \lceil Match \rceil \ \ \lceil Matches \rceil$
$(Match)$	$\lceil <$ **Match MatchID =** $MatchId >$ $<$ **AttrValue** $> value </$ **AttrValue** $>$ $<$ **AttributeDesignator** $ADHeader$ **/** $>$ $</$ **Match** $> \rceil$ $= MatchId(\lceil ADHeader \rceil, value)$
$(ADHeader)$	$\lceil$ **Category=** $Subject$ **AttributeId =** $AttSubject$ **DataType=** $type$ **MustBePresent=** $boolean \rceil = AttSubject$ $\lceil$ **Category= resource AttributeId =** $AttResource$ **DataType=** $type$ **MustBePresent=** $boolean \rceil = AttResource$ $\lceil$ **Category= action AttributeId =** $AttAction$ **DataType=** $type$ **MustBePresent=** $boolean \rceil = AttAction$ $\lceil$ **Category= environment AttributeId =** $AttEnv$ **DataType=** $type$ **MustBePresent=** $boolean \rceil = AttEnv$
$(Condition)$	$\lceil <$ **Condition** $> BooleanExpression </$ **Condition** $> \rceil = BooleanExpression$

3.5.5. Deny-unless-permit

Deny-unless-permit between $P_1, \ldots, P_n$ is formalized in SePL as follows:

$$DUP(P_1, \ldots, P_n) \approx \ulcorner(P_1 \llbracket \ldots \llbracket P_n)$$

3.5.6. Permit-unless-deny

Permit-unless-deny between $P_1, \ldots, P_n$ is formalized in SePL as follows:

$$DUP(P_1, \ldots, P_n) \approx \ulcorner(P_1 \rrbracket \ldots \rrbracket P_n)$$

3.6. Some Benefits of XACML Formalization

It is a known fact that formalization removes ambiguity from the textual description of semantics and opens the door to a variety of automatic analysis. Formalization transforms XACML policies to mathematical objects and allows to extract their properties (complete, conflicts, etc.) and their interrelationship (include, disjoint, etc.). Here are some useful applications:

- Simplifying security policies: Given a security policy P, we want to simplify it as much as possible. If a security policy is formalized as a combination of boolean expressions, as in SePL, many techniques allowing their simplifications exist. These kinds of simplifications can be useful to reduce the time of verifying complex security policies.

- Detecting conflicts and redundancies: Given two security policies P and Q, we want to compare them as it is reported in [13], i.e, P is equivalent to Q, P is included in Q, P and Q are disjoint, etc. This kind of analysis can be used to detect a conflict between rules (when some of them authorize access denied by others). For instance, if one rule allows an employee to get access into the system between 8am and 7pm and another denies the access between 5am and 8am, then there is a conflict. In fact, if the combining algorithm is "only-one-applicable", then the result can be indeterminate. It is preferable to detect this kind of overlap and show it to the end-user since they usually correspond to design errors in security policies. For the SePL language, since security policies are transformed into boolean expressions, different kinds of comparisons become easy. For instance, to compare whether P_1 and P_2 are equivalent or not, it is enough to check whether the expression $(P_1 \Rightarrow P_2) \wedge (P_2 \Rightarrow P_1)$ can be proved.

- Quantify distances between security policies: Given two security policies P and Q, we want to measure the distance between them. Many works like [16] and [22] have already addressed this problem. Our absolute semantics is helpful to define metrics allowing to measure distance between security policies. In fact, this semantics transforms a security policy to a domain (acceptance domain, reject domain). Therefore, we can use the Euclidean distance or other metrics in these domains to quantify similarity between them.

  We have also developed, using PHP and XML, a web based application allowing to automatically measure distances between XACML policies, transformed to SPL , using a variety of metrics as shown by Figure 3.3.

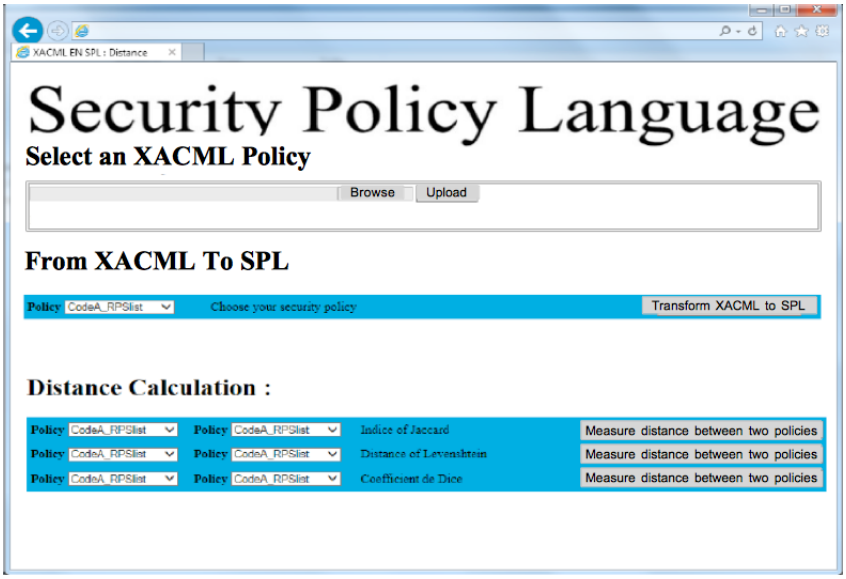

Figure 3.3: Measuring distances between XACML policies based on SPL

- Detecting incompleteness in security policy: Since our absolute semantics gives to any security policy P a pair (A, D) as semantics where A is the domain of accept and D the domain of deny, it is easy to extract the incompleteness domain of P which is $I_{\mathcal{D}} = \mathcal{D} - (A \cup D)$. This $I_{\mathcal{D}}$ shows the part of the domain where P is mute. If P is the whole security policy, then it is incomplete. In fact, a good security policy should have $I_{\mathcal{D}} = \emptyset$ (i.e. any action should be either denied or authorized). If for example, there is a rule stating that an employee can get access into the system between 8am to 5pm and a rule stating that the same employee cannot get access into the system between in [7pm, 8am], then it is incomplete since we don't know whether the employee is authorized or not in]5am, 7am[. Let P be a security policy such that its relative semantics in SePL is (B_P^+, B_P^-) where B_P^+ and B_P^- are two boolean expressions capturing the acceptance and the denial part of P. We can detect whether P is complete or not by verifying that $B_P^+ \vee B_P^-$ is always true or not.

- Using MTBDD techniques and tools: Since our relative semantics transforms XACML policies into boolean expressions where variables are elementary conditions, we can proceed like in [23] or in [12] to produce Multi-Terminal Binary Decision Diagrams (MTBDD). This MTBDD is useful to quickly analyze security policies and compare them. For instance, suppose that $[\![P]\!]_\Gamma = (a1 \wedge a2 \wedge a3, \neg a1 \wedge a2 \wedge a3 \wedge a4)$, where $a1$ is "$role = admin$", $a2$ is "$file = pwd$", $a3$ is "$access = write$" and $a4$ is "$role = student$". An MTBDD for P can be as shown in Figure 3.4. The reader can refer to [10] to know how we can produce a MTBDD with optimal size.

  For MTBDDs, we can answer YES or NO questions like is it possible that "student" can

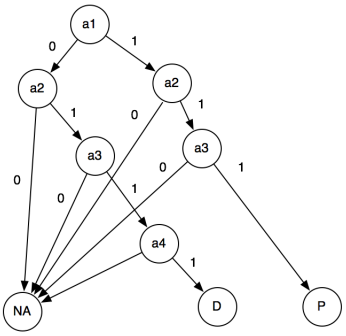

Figure 3.4: MTBDD Associated to XACML Policy

"write" in the "pwd" file. We can answer queries such as who has access to "pwd" file and we can compare two policies. A tool like Margave [14] is suitable to make this analysis. Combining MTBDDs related to two security policies can reveal similarity and dissimilarity between them as shown in Figure 3.5, where leafs having two similar labels show similarity and leafs having two different labels denote dissimilarity. A tool like Exam [22] can do this kind of analysis.

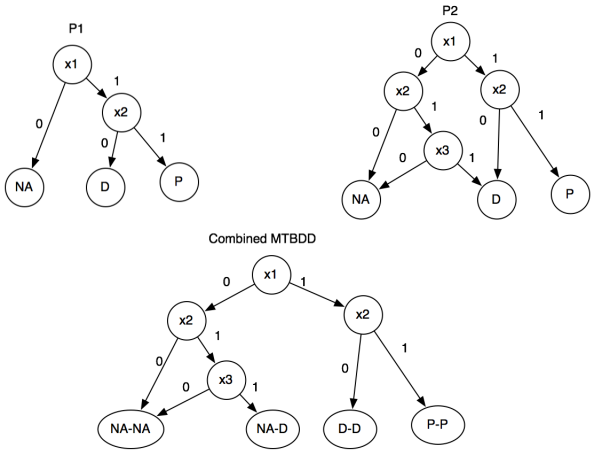

Figure 3.5: Combining MTBDD

Other interesting formalizations and analyses are also possible. For instance, Bryans et al.[11] formalize a fragment of XACML by translating it to the process algebra CSP [4]. This gives the possibility of using model checkers such as FDR to verify properties of security policies and to compare them to each others.

4. Completeness of SePL

Theorem 4.1. *The language is complete with respect to set theory according to the following meaning: let P and P' be two policies such that $[\![P]\!] = (A, D)$ and $[\![P']\!] = (A', D')$, we can build using the operator of the SePL a policy where the semantics has the form $(f(A, D, A', D'), g((A, D, A', D'))$ where f and g are built based on the combination of $\cup$, $\cap$, $-$ and complement and $f(A, D, A', D')$ and $g(A, D, A', D')$ are disjoint. The importance of such result is that it gives a lower bound for the expressiveness of SePL that seems to be enough for capturing almost all combining algorithms of XACML since they are related to combining sets (accepting and denying sets).*

Proof. If we are able to build the policies F such that $[\![F]\!] = (f(A, D, A', D'), \perp)$ and G such that $[\![G]\!] = (\perp, g((A, D, A', D'))$. Then, since $f(A, D, A', D')$ and $g((A, D, A', D')$ are disjoint, we deduce that

$$[\![F + G]\!] = (f(A, D, A', D'), g((A, D, A', D'))$$

We notice also that if we are able to build F, it is immediate that we can build G. Now, let us prove that we can build F. We do the proof by induction on the size of F. We show that we can build F if it contains zero operator then we suppose that we can build it for any size lower or equal than n and prove that that we can build it for a size $n + 1$.

- $n = 0$: In this case $f(A, D, A', D')$ has one of the following forms:
  - $f(A, D, A', D') = A$, in this case it is immediate that $F = P||1$
  - $f(A, D, A', D') = D$, in this case it is immediate that $F = (\neg P)||1$
  - $f(A, D, A', D') = A'$, in this case it is immediate that $F = P'||1$
  - $f(A, D, A', D') = D'$, in this case it is immediate that $F = (\neg P')||1$
- $n + 1$: In this case there exist f_1 and f_2 such that $size(f_1) \leq n$, $size(f_2) \leq n$ and $f(A, D, A', D')$ has one of the following forms:
  - $f(A, D, A', D') = f_1(A, D, A', D') \cup f_2(A, D, A', D')$

    By induction, it follows that there exist F_1 and F_2 such that $[\![F_1]\!] = (f_1((A, D, A', D'), \perp)$ and $[\![F_2]\!] = (f_2((A, D, A', D'), \perp)$. It follows from the definition of the semantics of $[\![(P + Q)]\!]$ that $[\![F_1 + F_2]\!] = (f(A, D, A', D'), \perp)$ and we conclude that

    $$F = F_1 + F_2$$
  - $f(A, D, A', D') = f_1(A, D, A', D') \cap f_2(A, D, A', D')$

By induction, it follows that there exist F_1 and F_2 such that $[\![F_1]\!] = (f_1((A, D, A', D'), \bot)$ and $[\![F_2]\!] = (f_2((A, D, A', D'), \bot)$. It follows from the definition of the semantics of $[\![(P||Q)]\!]$ that $[\![F_1||F_2]\!] = (f(A, D, A', D'), \bot)$ and we conclude that

$$F = F_1||F_2$$

- $f(A, D, A', D') = f_1(A, D, A', D') - f_2(A, D, A', D')$

  By induction, it follows that there exist F_1 and F_2 such that $[\![F_1]\!] = (f_1((A, D, A', D'), \bot)$ and $[\![F_2]\!] = (f_2((A, D, A', D'), \bot)$. It follows from the definition of the semantics of $[\![(P + Q)]\!]$ that $[\![F_1 + \neg F_2]\!] = (f_1(A, D, A', D') - f_2(A, D, A', D'), f_2(A, D, A', D') - f_1(A, D, A', D'))$ and we conclude that

$$F = (F_1 + \neg F_2)||1$$

- $f(A, D, A', D') = \overline{f_1(A, D, A', D')}$

  By induction, it follows that there exist F_1 such that $[\![F_1]\!] = (f_1((A, D, A', D'), \bot)$. It follows from the definition of the semantics of $[\![(P + Q)]\!]$ that $[\![1 + \neg F_1]\!] = (\mathcal{D} - f_1(A, D, A', D'), f_1(A, D, A', D') - \mathcal{D}) = (\overline{f_1(A, D, A', D')}, \bot)$ and we conclude that

$$F = 1 + \neg F_1$$

□

Corollary 4.2. *Let P and P' two policies such that $[\![P]\!] = (A, D)$ and $[\![P']\!] = (A', D')$. The previous theorem gives an algorithm that allows to construct Q such that*

$$[\![Q]\!] = (f(A, D, A', D'), g(A, D, A', D'))$$

The algorithm is as follows:

1: ***function*** Semantics2Policy
2: ***Input:*** (A, D) , (A', D') *and* (f, g)
3: ***Output:*** *A policy Q such* $[\![Q]\!] = (f, g)$
4: ***return*** S2P$((A, D), (A', D'), f) + \neg($S2P$((A, D), (A', D'), g))$
5: ***end function***

The function S2P *is defined as follows:*

1: ***function*** S2P
2: ***Input:*** (A, D), (A', D') *and* f
3: ***Switch***
4: *case A:* ***return*** $P||1$

5: *case* D: ***return*** $\neg P||1$
6: *case* A': ***return*** $P'||1$
7: *case* D': ***return*** $\neg P'||1$
8: *case* $f_1 \cup f_2$: ***return*** $\mathsf{S2P}((A,D),(A',D'),f_1) + \mathsf{S2P}((A,D),(A',D'),f_2)$
9: *case* $f_1 \cap f_2$: ***return*** $\mathsf{S2P}((A,D),(A',D'),f_1) \,||\, \mathsf{S2P}((A,D),(A',D'),f_2)$
10: *case* $f_1 - f_2$: ***return*** $(\mathsf{S2P}((A,D),(A',D'),f_1) + (\neg(\mathsf{S2P}((A,D),(A',D'),f_2))) \,||\, 1)$
11: *case* $\overline{f_1}$: ***return*** $1 + \neg(\mathsf{S2P}((A,D),(A',D'),f_1))$
12: ***end function***

Proof. It is immediate from the constructive proof of Theorem 4.1 □

5. Simplifying SePL

Definition 5.1. *Let P_1 and P_2 be two policies such that $[\![P_1]\!] = (A_1, D_1)$ and $[\![P_2]\!] = (A_2, D_2)$. We say that:*

- *P_1 is lower than P_2, denoted by $P_1 \sqsubseteq P_2$ if $A_1 \subseteq A_2$ and $D_1 \subseteq D_2$.*
- *P_1 is equivalent to P_2, denoted by $P_1 \approx P_2$ if $P_1 \sqsubseteq P_2$ and $P_2 \sqsubseteq P_1$.*

Proposition 5.2. *Let $\mathsf{P_1}$, $\mathsf{P_2}$ and $\mathsf{P_3}$ be three policies. We have:*

$$\mathsf{P_1} \Vert \mathsf{P_2} \approx \neg(\neg\mathsf{P_1} \Vert \neg\mathsf{P_2})$$

Proof.

$$
\begin{aligned}
& [\![\neg(\neg\mathsf{P_1} \Vert \neg\mathsf{P_2})]\!] \\
= \;& \{\!| \text{ Definition of } [\![\neg(P)]\!] |\!\} \\
& \overline{[\![\neg\mathsf{P_1} \Vert \neg\mathsf{P_2}]\!]} \\
= \;& \{\!| \text{ Definition of } [\![\mathsf{P} \Vert \mathsf{Q}]\!] |\!\} \\
& \overline{((D_1 \cup D_2, (A_1 - D_2) \cup (A_2 - D_1))} \\
= \;& \{\!| \text{ Definition of } \overline{(\alpha,\beta)} |\!\} \\
& (((A_1 - D_2) \cup (A_2 - D_1), D_1 \cup D_2) \\
= \;& \{\!| \text{ Definition of } [\![\mathsf{P} \Vert \mathsf{Q}]\!] |\!\} \\
& [\![\mathsf{P_1} \Vert \mathsf{P_2}]\!]
\end{aligned}
$$

□

Proposition 5.3. *Let* P_1, P_2 *and* P_3 *be three policies. We have the following results:*

1. $\mathsf{P}_1 + \mathsf{P}_2 \approx \mathsf{P}_2 + \mathsf{P}_1)$
2. $\mathsf{P}_1 + (\mathsf{P}_2) + \mathsf{P}_3) \approx (\mathsf{P}_1 + \mathsf{P}_2) + \mathsf{P}_3$
3. $\mathsf{P}_1||\mathsf{P}_2 \approx \mathsf{P}_2||\mathsf{P}_1$
4. $\mathsf{P}_1||(\mathsf{P}_2)||\mathsf{P}_3) \approx (\mathsf{P}_1||\mathsf{P}_2)||\mathsf{P}_3$
5. $(\mathsf{P}_1 + \mathsf{P}_2)||\mathsf{P}_3 \approx \mathsf{P}_1||\mathsf{P}_3 + \mathsf{P}_2||\mathsf{P}_3$
6. $\neg 1 \approx 0$
7. $\neg 0 \approx 1$
8. $\neg\varepsilon \approx \varepsilon$
9. $\neg(\neg\mathsf{P}_1) \approx \mathsf{P}_1$
10. $\neg(\mathsf{P}_1||\mathsf{P}_2) \approx \neg\mathsf{P}_1||\neg\mathsf{P}_2$
11. $\neg(\mathsf{P}_1 + \mathsf{P}_2) \approx \neg\mathsf{P}_1 + \neg\mathsf{P}_2$
12. $\mathsf{P}_1 + \neg\mathsf{P}_2 \approx \varepsilon$
13. $\mathsf{P}_1 + \varepsilon \approx \mathsf{P}_1$
14. $\mathsf{P}_1||\neg\mathsf{P}_2 \approx \varepsilon$

Proof. Most of them are immediate from the definition of the semantics □

Theorem 5.4. *The following language has the same expressivity as SePL:*

$$\begin{array}{rcl} \mathsf{P}, \mathsf{P}_1, \mathsf{P}_2 & ::= & \mathsf{R} \,|\, \neg P \,|\, \ulcorner P \,|\, \mathsf{P}_1 \parallel \mathsf{P}_2 \\ & & |\, \mathsf{P}_1 + \mathsf{P}_2 \,|\, \mathsf{P}_1 \ominus \mathsf{P}_2 \\ \mathsf{R} & ::= & < \varphi_1, \varphi_2 > \end{array}$$

Proof.

- Since $[\![\ P - P'\ ]\!] = (A - (A' \cup D'), D - (A' \cup D'))$, then from Corollary 4.2 we have: $P - P' \approx (P||1 + \neg(P'||1 + \neg P'||1))||1 + \neg((\neg P||1 + \neg(P'||1 + \neg P'||1))||1)$
- Since $[\![\ P.P'\ ]\!] = (A \cup (A' - D), D \cup (D' - A))$, then from Corollary 4.2 we have: $P.P' \approx P||1 + (P'||1 + \neg(\neg P||1))||1 + \neg(\neg P||1 + (\neg P'||1 + \neg(P||1)||1))$
- Since $[\![\ P \| P'\ ]\!] = (A \cup A', (D - A') \cup (D' - A))$, then from Corollary 4.2 we have: $P.P' \approx P||1 + P'||1 + \neg((\neg P||1 + \neg(P'||1))||1 + (\neg P'||1 + \neg(P||1))||1)$

□

6. Comparison with the Related Work

The standard language XACML is increasingly used in a wide variety of domains. A security policy can involve a big number of rules combined using different algorithms. However, since the semantics of XACML is described using the natural language, it will be difficult and time consuming for humans to get its meaning and analyze it. For that reason, many formal languages have been defined during the few past years and used to capture and analyze security policies specified in XACML or other security policy languages. We review in what follows these research initiatives and pinpoint the differences with our work.

$\mathcal{D}$-algebra. In [19], Ni et al. define the elegant $\mathcal{D}$-algebra as follows:

Definition 6.1 ($\mathcal{D}$-Algebra)**.** *Let $\mathcal{D}$ be a nonempty set of elements, 0 be a constant element of $\mathcal{D}$, $\neg$ be a unary operation on element in $\mathcal{D}$, and $\oplus^{\mathcal{D}}$, $\otimes^{\mathcal{D}}$ be binary operations on element in $\mathcal{D}$. A $\mathcal{D}$-Algebra is an algebraic structure $(\mathcal{D}, \neg, \oplus^{\mathcal{D}}, \otimes^{\mathcal{D}}, 0)$ closed on $\neg$, $\oplus^{\mathcal{D}}$, $\otimes^{\mathcal{D}}$ and satisfying the following axioms:*

1. $x \oplus^{\mathcal{D}} y = y \oplus^{\mathcal{D}} x$
2. $(x \oplus^{\mathcal{D}} y) \oplus^{\mathcal{D}} z = x \oplus^{\mathcal{D}} (y \oplus^{\mathcal{D}} z)$
3. $x \oplus^{\mathcal{D}} 0 = x$
4. $\neg\neg x = x$
5. $x \oplus^{\mathcal{D}} \neg 0 = \neg 0$
6. $\neg(\neg x \oplus^{\mathcal{D}} y) \oplus^{\mathcal{D}} y = \neg(\neg y \oplus^{\mathcal{D}} x) \oplus^{\mathcal{D}} x$
7. $x \otimes^{\mathcal{D}} y = \begin{cases} \neg 0 & : x = y \\ 0 & : x \neq y \end{cases}$

The interpretation of $\mathcal{D}$-Algebra on XACML decisions is as follows:

- $\mathcal{D}$ is $\mathcal{P}(\{p, d, \frac{n}{a}\})$, where $\emptyset$ is the empty policy, $\{p\}$ is permit, $\{d\}$ is deny, $\{\frac{n}{a}\}$ is not applicable, $\{p, d\}$ is conflict, $\{p, \frac{n}{a}\}$ is indeterminate permit, $\{d, \frac{n}{a}\}$ is indeterminate deny and $\{p, d, \frac{n}{a}\}$ is indeterminate permit-deny.
- 0 is the empty set.
- $\neg x$ is $(\{p, d, \frac{n}{a}\} - x)$.
- $x \oplus^{\mathcal{D}} y$ is $x \cup y$.
- $x \otimes^{\mathcal{D}} y$ is defined by axiom 7.

The formalization of permit-overrides of two policies x and y in $\mathcal{D}$-Algebra is given by $f_{po}(x, y)$ as follows:

$$\begin{aligned} fpo(x,y) &= (x \oplus^{\mathcal{D}} y) \\ &= \ominus^{\mathcal{D}}(((x \otimes^{\mathcal{D}} \{p\}) \oplus^{\mathcal{D}} (y \otimes^{\mathcal{D}} \{p\})) \\ &\quad \odot^{\mathcal{D}} \{d, \frac{n}{a}\}) \\ &= \ominus^{\mathcal{D}}(\neg((x \odot^{\mathcal{D}} y) \otimes^{\mathcal{D}} \{\frac{n}{a}\}) \odot^{\mathcal{D}} \{\frac{n}{a}\} \\ &\quad \odot^{\mathcal{D}} \neg((x \times^{\mathcal{D}} \emptyset) \oplus^{\mathcal{D}} (y \times^{\mathcal{D}})) \end{aligned}$$

where $(x \odot^{\mathcal{D}} y)$ and $(x \ominus^{\mathcal{D}} y)$ are shortcuts for $\neg(\neg x \oplus^{\mathcal{D}} \neg y)$ and $(x \odot^{\mathcal{D}} \neg y)$ respectively.

$\mathcal{D}$-Algebra is neither conform with XACML-3.0 nor with XACML-2.0. As in [25], we show in Table 7 the result of applying the "permit-overrides" algorithm for composing $P1 = \{p, \frac{n}{a}\}$ (Indeterminate Permit) and $P2 = \{d\}$ (Deny). The result expected by XACML-3.0 is "Indeterminate Permit-Deny", however $\mathcal{D}$-Algebra returns *conflict*.

PBel Logic. In [17, 18], the authors use a four value logic (*grant*, *deny*, *conflict*, *undefined*) called PBel and it is derived from the Blenap Logic [3]. They also define a query language allowing to specify questions related to policy analysis such as conflict freedom and gap freedom policies. Queries are finally transformed into a fragment of the first order logic for which satisfiability and validity checks can be done by SAT solvers or BDDs.

The syntax of PBel is as shown in the following BNF grammar:

p, q	$::=$	Policy
	b if ap_i	Basic policy
	$\neg p$	Logical negation
	$p \wedge q$	Logical meet
	$p \supset q$	Implication
	$p \oplus q$	Nondeterministic choice
	$p[v \mapsto q]$	Refinement

where ap_i are access predicates, b is either tt (*permit*) of ff (*deny*) and v is either $\bot$ (*non-applicable*) or $\top$ (*indeterminate*, i.e., both *permit* and *deny* arise). $p \oplus q$ grants (respectively denies) an access if at least one accepts (receptively denies) and the other is either *accept* (respectively *deny*) or *not applicable*. $p[v \mapsto q]$ yields p if $p \neq v$ and q otherwise. Using this semantics, the authors formalize the "permit-overrides" algorithm as follows: $(p \oplus q)[\top \mapsto ff]$. However, as shown in [25], this formalization is not consistent with XACML. In fact, the result of "permit-overrides" algorithm for composing $P1 = \top$ (*Indeterminate*) and $P2 = ff$ (*deny*) should be, according to XACML, *indeterminate* and not *deny* as returned by this logic.

XACML Logic. In [25], the authors proposed an XACML logic having the syntax shown in Table 6:

- The semantics of *Match*, *Target* and *Condition* is given by a function $[\![-]\!]$ returning results in the three-valued lattice $V_3(\{\top, I, \bot\}, \leq)$, where $\top \leq I \leq \bot$ and $\top$ means match (also

Table 6: Syntax of the Logic of XACML [25]

$PolicySet$	::=	$< Target, < PolicySet_1, ..., PolicySet_m >, \theta >$	
		$\mid < Target, hPolicy_1, ..., Policy_m >, \theta >$	where $m \geq 0$
$Policy$	::=	$< Target, < Rule_1, ..., Rule_m >,$	where $m \geq 1$
$Rule$	::=	$< Effect, Target, Condition >$	
$Condition$	::=	propositional formulae	
$Target:$	:=	$Null$	
		$\mid AnyOf_1 \wedge ... \wedge AnyOf_m$	where $m \geq 1$
$AnyOf$	::=	$AllOf_1 \vee ... \vee AllOf_m$	where $m \geq 1$
$AllOf$	::=	$Match_1 \wedge ... \wedge Match_m$	where $m \geq 1$
$Match$	::=	$\Phi(\alpha)$	
Φ	::=	subject |action |resource |enviroment	
α	::=	attribute value	
θ	::=	$p-o \mid d-o \mid f-a \mid o-1-a$	
$Effect$	::=	$d \mid p$	
		XACML Request Component	
$Request$	::=	$A1, ..., Am$	where $m \geq 1$
A	::=	$\Phi(\alpha)$ | external state	

means *true* or *applicable*), $\perp$ means not match (also means *false* or not *applicable*) and I means *indeterminate*.

- The semantics of a *Match* $\mathcal{M}$ for a given query $\mathcal{Q}$, denoted by $[\mathcal{M}](\mathcal{Q})$ is $\top$ if $\mathcal{M} \in \mathcal{Q}$, $\perp$ if $\mathcal{M} \notin \mathcal{Q}$ and I if there is an error during evaluation.

- The semantics of a *Condition* $\mathcal{C}$ for a given query $\mathcal{Q}$ is $eval(\mathcal{C}, \mathcal{Q})$, where $eval$ is a given function that evaluates conditions.

- The semantics of composed elements using $\wedge$ such that $[\mathcal{E}_1 \wedge \ldots \wedge \mathcal{E}_n](\mathcal{Q})$, where $\mathcal{E}_1, \ldots, \mathcal{E}_n$ are $Target$s or $AllOfs$, is $[\mathcal{E}_1](\mathcal{Q}) \sqcap \ldots \sqcap [\mathcal{E}_n](\mathcal{Q})$ where $\sqcap$ gives the least upper bound according to the lattice V_3.

- The semantics of composed elements using $\vee$ such that $[\mathcal{E}_1 \vee \ldots \vee \mathcal{E}_n](\mathcal{Q})$, where $\mathcal{E}_1, \ldots, \mathcal{E}_n$ $AnyOfs$, is $[\mathcal{E}_1](\mathcal{Q}) \sqcup \ldots \sqcup [\mathcal{E}_n](\mathcal{Q})$ where $\sqcup$ gives the greatest lower bound according to the lattice V_3.

- To give the semantics of a *Rule*, the authors extended V_3 to $V_6 = \{\top_p, \top_d, I_p, I_d, I_{pd}, \perp\}$ where $\top_p$ is *Permit*, $\top_d$ is *Deny*, I_p is *Indeterminate Permit*, I_d is *Indeterminate Deny*, I_{pd} is *Indeterminate Permit-Deny*, and $\perp$ is *Not match*.

- The semantics of a *Rule* $\mathcal{R} =< *, \mathcal{T}, \mathcal{C} >$, where $*$ is either p or d, is as follows:

Figure 6.1: $\mathcal{L}_{p-o}$

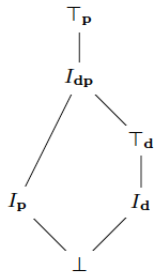

Table 7: Consistency of various logics with respect to XACML

Logic	P_1	P_2	Permit-Overrides Algorithm	*Result*	*Consistancy*
Belnap logic	$\top$	ff	$(\top \oplus ff)(\top \mapsto ff)$	ff	–
$\mathcal{D} - Algebra$	$\{p, \frac{n}{a}\}$	$\{d\}$	$f_{po}(\{p, \frac{n}{a}\}, \{d\})$	$\{p, d\}$	–
$V6$	I_p	$\top_d$	$\bigoplus_{p-o}(< I_d, \top_d >)$	I_{dp}	$\surd$
P	$[0, \frac{1}{2}]$	$[0, 1]$	$\bigoplus_{p-o}(< [0, \frac{1}{2}], [0, 1] >)$	$[\frac{1}{2}, \frac{1}{2}]$	$\surd$
$SePL$	$(?, \mathsf{F})$	$(\mathsf{F}, \mathsf{T}) \vee (?, \mathsf{T})$	$(?, \mathsf{F}) \oplus ((\mathsf{F}, \mathsf{T})) \vee (?, \mathsf{T})$	$(?, ?)$	$\surd$

$$[\![\mathcal{R}]\!](\mathcal{Q}) = \begin{cases} \top_* & [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = \top \\ \bot_* & ([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = \bot) \\ & \text{or } [\![\mathcal{T}]\!](\mathcal{Q}) = \bot \\ I_* & \text{otherwise} \end{cases}$$

- For the combining algorithms "p_o", "d_o", "f_a" and "$o - 1 - a$", different lattices and combining rules are used to define the semantics. For the p_o for example, the lattice $\mathcal{L}_{p-o}$ of Figure 6.1 is used. For a sequence $S =< P_1, \ldots, P_n >$ of policies, the permit override of S, denoted by $\bigoplus_{p-o}(S)$, is $\sqcup_{p-o}\{s_1, \ldots, s_n\}$, where s_i is the semantics of P_i and $\sqcup_{p-o}$ is the greatest lower bound according to the lattice $\mathcal{L}_{p-o}$.

- The authors defined also an equivalent semantics to V_6, called P, based on the set $\{[0,0], [\frac{1}{2}, 0], [0, \frac{1}{2}], [\frac{1}{2}, \frac{1}{2}], [1,0], [0,1]\}$, where $[0,0]$ is equivalent to $\bot$, $[\frac{1}{2}, 0]$ is equivalent to I_d, $[0, \frac{1}{2}]$ is equivalent to I_p, $[\frac{1}{2}, \frac{1}{2}]$ is equivalent to I_{dp}, $[1,0]$ is equivalent to $\top_d$ and $[0,1]$ is equivalent to $\top_p$.

Fine-grained Integration Algebra (FIA). In [23], Rao et al. proposed a Fine-grained Integration Algebra, denoted by FIA, for policy integration. It is a three-valued and language-independent algebra defined by the following element $(\Sigma, PY, PN, +, \&, \neg, \Pi_{dc})$, where Σ is a vocabulary of attribute names and their domains, PY and PN are two policy constants, $+$ and $\&$ are two binary operators, and $\neg$ and Π_{dc} are two unary operators. The semantics of a policy P can be considered as a 2-tuple (R_Y^P, R_N^P) where R_Y^P and R_N^P are the sets of requests that are permitted and denied by P respectively, and $R_Y^P \cap R_N^P = \emptyset$. A policy P is considered as not applicable, if the request is not in $R_Y^P \cup R_N^P$. The semantics of P can also be viewed as a function mapping each request to a value in $\{Y, N, NA\}$. A request r is a set of pairs $\{(a_1, v_1), (a_2, v_2), \ldots, (a_n, v_n)\}$ where, for all $i \in \{1, \ldots, n\}$, a_i is an attribute name and v_i is a value in the domain of a_i, e. g. $r = \{(role, manager), (act, read), (time, 10am)\}$. PY is a policy constant that permits everything whereas PN is the one that denies everything. $P1 + P2 = (R_Y^{P1} \cup R_Y^{P2}, (R_N^{P1} - R_Y^{P1}) \cup (R_N^{P2} - R_Y^{P2}))$, $P1 \& P2 = (R_Y^{P1} \cap R_Y^{P2}, R_N^{P1} \cap R^{P2})$, $\neg P = (P_N, P_Y)$, $\Pi_{dc}(P) = (\{r \in P_Y \text{ and } r \text{ satisfies } dc\}, \{r \in P_N \text{ and } r \text{ satisfies } dc\})$. The authors used FIA to express the following XACML policy combining algorithms: Permit-overrides, Deny-overrides, First-one-applicable, Only-one-applicable. For instance, the combination of policies $P_1, \ldots, P_n$ under the algorithm Permit-overrides is expressed by $P_1 + \ldots + P_n$. Nevertheless, no one of these combination algorithms is XACML compatible since the algebra does not deal with indeterminate situation such that Indeterminate Permit and Indeterminate Deny. Furthermore, the authors do not give any translation function from XACML to FIA.

Defeasible Description Logic (DDL). In [15], Kolovski et al. used an extended version of Description Logic called Defeasible Description Logic (DDL). The authors succeed to transform the main concepts of XACML (attribute, value, effect, Rule, Policy) to DDL in order to make different formal analysis such as policy comparison and verification. They claimed that their XACML formalization supports Permit-Overrides, Deny-Overrides and First-Applicable combining algorithms. However, there XACML formalizations are not compatible with XACML since they do not handle indeterminate cases like Indeterminate Permit, Indeterminate Deny and Indeterminate Permit-Deny.

Table 8: Comparison with the related work

Features/Languages	Belnap Logic	$\mathcal{D}-Algebra$	$V6$	P	DDL	FIA	$SePL$
The logic addresses explicitly XACML-3.0			√	√			√
The compatibility with XACML algorithms has been proved							√
The language is generic: syntax and sematics are independent from XACML	√	√			√	√	√
Logic is endowed with a semantics that is independent from any evaluaion model						√	√

Advantages of using SePL. Here, we compare the previous logics to SePL. Table 8 summarizes this comparison.

- Contrarily to the XACML logic, SePL, Belnap-Logic and $\mathcal{D}-Algebra$ are independent from XACML. This means that if we want to formalize a larger subset of XACML, we do not necessarily need to extend the language itself. Besides, other policy languages, especially those used to specify firewall policies, can also be formalized using languages like SePL, Belnap-Logic and $\mathcal{D}-Algebra$.

- The compatibility of SePL has been formally proved for almost all the combining algorithms. For Belnap Logic and D-Algebra, there is a proof that they are not compatible and for $V6$ and P there is neither proof of compatibility nor a proof of incompatibility for all combining algorithms.

- Like $V6$ and P, SePL addresses the recent version XACML-3.0. However, SePL formalizes a larger fragment of XACML-3.0 than $V6$ and P and it includes most of combining algorithms.

- The SePL logic is endowed with an absolute semantics based on set theory allowing to understand the meaning of operators and policies and to analyze them outside any interpretation model. Another interesting application of this semantics is that it allows easily quantify distance between XACML policies. In fact, the absolute semantics of a policy is a domain (a subsets of $\mathcal{D}$) so measuring distances between policies is equivalent to measuring distance between domains and there are many distance metrics allowing that such as Euclidean distance.

- Many Multi-valued logics (3-valued, 4-valued, 6-valued) have been used to formalize XACML semantics. However, it was not always clearly justified how many values we need and why. This is not the case for SePL, since the semantics of policy is a pair where each element has 3 possible values (T, F or ?), we obtain an 8-valued logic ((T,T), (T,F), $\{(\mathsf{T},?)$, (F,T), $(\mathsf{F},?)$, $(?,?)\}$ allowing to easily distinguish the six different scenarios of XACML-3.0 ($Pemit$, $Deny$, $IndeterminatePermit$, $IndeterminateDeny$, $IndeterminatePermitDeny$ and $NotApplicable$).

7. Conclusion

We presented in this paper a simple, formal, and compact security policy language called SePL. We have shown how complex real-world security policy languages, such as XACML-3.0 can be captured in a simple and understandable way using SePL. Furthermore, we proved the completeness of SePL with respect to set theory and we proposed a tool allowing to automatically transform XACML policies to SPL and measure distances between them.

Our future work includes the extension of the tool so that we can simplify security policies, detect their conflict, redundancy and incompleteness, and integrate the MTBDD techniques to answer more questions about them.

References

References

[1] J. Lukasiewicz. On Three-Valued Logic. In: Jan Lukasiewicz Selected Works. (L. Borkowski, ed.), North Holland, p. 87-88, 1920.

[2] S. C. Kleene. Introduction to Metamathematics. North-Holland, 1952.

[3] N. D. Belnap. A useful four-valued logic. In G. Epstein and J.M. Dunn, editors, Modern Uses of Multiple-Valued Logic, p. 8-37, D. Reidel, Dordrecht, 1977.

[4] C. A. R. Hoare. Communicating Sequential Processes Commun. ACM, 21(8), p. 666-677, 1978.

[5] M. Fitting. A Kripke-Kleene semantics for logic programs. Journal of Logic Pro- gramming, 2(4), p. 295-312, 1985.

[6] J. R. Levine, T. Mason, D. Brown. Lex & Yacc 2nd edition. O'Reilly & Associates. ISBN: 1565920007, 1992.

[7] J. Lobo, N. Dulay, R. Bhatia, and S. Naqvi. A Policy Description Language, In Proceedings of of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence, p. 291-298, 1999.

[8] S. Hada and M. Kudo. XML Access Control Language: Provisional Authorization for XML Documents. `http://www.research.ibm.com/trl/projects/xml/xacl/xacl-spec.html`, 2000.

[9] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language, In Proceedings of the International Workshop on Policies for Distributed Systems and Networks, p. 18-38, 2001.

[10] O. Grumberg, S. Livne, and S. Markovitch. Learning to order bdd variables in verification. Journal of Artificial Intelligence Research, 18, p. 83-116, 2003.

[11] J. Bryans. Reasoning About XACML Policies Using CSP. In Proceedings of the 2005 Workshop on Secure Web Services, p. 28-35 (SWS '05), 2005.

[12] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies, ICSE, p. 196-205, 2005.

[13] P. Mazzoleni, E. Bertino, and B. Crispo. Xacml policy integration algorithms. In Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT), p. 223-232, 2006.

[14] D. Lin, P. Rao, E. Bertino, and J. Lobo. An approach to evaluate policy similarity. In Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT), p. 1-10, 2007.

[15] V. Kolovski, J. Hendler and B. Parsia. Formalizing XACML Using Defeasible Description Logics. In Proceedings of the 15th International World Wide Web conference (WWW'2007), p. 677-686, 2007.

[16] D. Lin, P. Rao, E. Bertino, and J. Lobo. An approach to evaluate policy similarity. In Proceedings of the ACM Symposium on Access Control Models and Technologies (SACMAT), p. 1-10, 2007.

[17] G. Bruns, D. S. Dantas and M. Huth. A simple and expressive semantic framework for policy composition in access control. in Proceedings of the 2007 ACM Workshop on Formal Methods in Security Engineering (FMSE'07), 2007.

[18] G. Bruns and M. Huth. Access-control via belnap logic: Effective and efficient composition and analysis. in 21st IEEE Computer Security Foundations Symposium (CSFS'08), 2008.

[19] Q. Ni, E. Bertino, and J. Lobo. D-algebra for composing access control policy decisions. In ASIACCS'09: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security, p. 298-309, 2009.

[20] OASIS. SAML Specifications. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security`, 2009.

[21] M. Migliavacca, I. Papagiannis, D. Eyers, B. Shand, J. Bacon, and P. Pietzuch. Distributed Middleware Enforcement of Event Flow Security Policy, In Proceedings of Middleware 2010, LNCS 6452, p 334-354, 2010.

[22] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. EXAM - a comprehensive environment for the analysis of access control policies. In International Journal of Information Security, 9(4), p. 253-273, 2010.

[23] P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo. Fine-grained integration of access control policies. Computers & Security 30(2-3), p. 91-107, 2011.

[24] M. Masi, R. Pugliese and F. Tiezzi Formalisation and Implementation of the XACML Access Control Mechanism. Engineering Secure Software and Systems, 7159, p. 60-74, 2012.

[25] C. D., P. K. Ramli, H. R. Nielson and F. Nielson, The Logic of XACML. Formal Aspects of Component Software, LNCS 7253, p. 205-222, 2012.

[26] C. D., P. K. Ramli, H. R. Nielson and F. Nielson, XACML 3.0 in Answer Set Programming. In Logic-Based Program Synthesis and Transformation, LNCS-7844, p. 89-105, 2013.

[27] OASIS. eXtensible Access Control Markup Language (XACML), OASIS Standard 3.0. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf, 2013.

Appendix

We provide in this part, the proofs of the conformance of our semantics for each XACML combining algorithm and its XACML specification.

7.1. Permit-override:

Permit-overrides between $P_1, \ldots, P_n$, denoted by $POR(P_1, \ldots, P_n)$: It accepts if at least one policy accepts and denies if no one accept and at least one denies. It can be formalized in SePL as follows:

$$POR(P_1, \ldots, P_n) \approx P_1 \Vert \ldots \Vert P_n$$

.

According to the XACML specification [27], we have:

"The permit overrides combining algorithm is intended for those cases where a permit decision should have priority over a deny decision. This algorithm has the following behavior.
1. If any decision is "Permit", the result is "Permit".
2. Otherwise, if any decision is "IndeterminateDP", the result is "IndeterminateDP".
3. Otherwise, if any decision is "IndeterminateP" and another decision is "IndeterminateD" or "Deny", the result is "IndeterminateDP".
4. Otherwise, if any decision is "IndeterminateP", the result is "IndeterminateP".
5. Otherwise, if any decision is "Deny", the result is "Deny".
6. Otherwise, if any decision is "IndeterminateD", the result is "IndeterminateD".
7. Otherwise, the result is "NotApplicable".
"

Let $[\![P_1]\!]_\Gamma = (a_1, d_1)$ $[\![P_2]\!]_\Gamma = (a_2, d_2)$, from the definition of relative semantics, we have:

$$[\![\mathsf{P}_1 \Vert \mathsf{P}_2]\!]_\Gamma \quad = \quad (a_1 \vee a_2, (d_1 - a_2) \vee (d_2 - a_1))$$

where $a - b$ is an abbreviation of $a \wedge \neg b$

Now let's see whether our semantics is compliant with the XACML specification.

1. Suppose that $[\![P_1]\!]_\Gamma = \mathsf{Permit}$ (The result is the same when we consider $[\![P_2]\!]_\Gamma = \mathsf{Permit}$ since the operator $\Vert$ is commutative). It means, from the definition of Permit, that $[\![P_1]\!]_\Gamma = (\mathsf{T}, d_1)$ where $d_1 \in \{\mathsf{F}, ?\}$. Suppose that $[\![P_2]\!]_\Gamma = (a_2, d_2)$. It follows that:

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_1 [\!]\!] \mathsf{P}_2\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_1 [\!]\!] \mathsf{P}_2\ ]\!]_\Gamma |\!\} \\
 & (\mathsf{T} \vee a_2, (d_1 - a_2) \vee (d_2 - \mathsf{T})) \\
= & \{\!|\text{From the truth tables } (\mathsf{T} \vee a_2) = \mathsf{T}|\!\} \\
 & (\mathsf{T}, (d_1 - a_2) \vee (d_2 - \mathsf{T})) \\
= & \{\!|a - b = a \wedge \neg b|\!\} \\
 & (\mathsf{T}, (d_1 \wedge \neg a_2) \vee (d_2 \wedge \neg \mathsf{T})) \\
= & \{\!|\text{From the truth tables}|\!\} \\
 & (\mathsf{T}, (d_1 \wedge \neg a_2)) \\
= & \{\!|\text{since } d_1 \in \{\mathsf{F}, ?\} \text{ then from the truth tables } (d_1 \wedge \neg a_2) \in \{\mathsf{F}, ?|\!\} \\
 & \mathsf{Permit}
\end{array}
$$

2. Suppose that $[\![\ P_1\ ]\!]_\Gamma = \mathsf{Indeterminate(PD)}$ (The result is the same when we consider $[\![\ P_2\ ]\!]_\Gamma = \mathsf{Indeterminate(PD)}$ since the operator $[\!]$ is commutative). It means from the definition of $\mathsf{Indeterminate(PD)}$, that $[\![\ P_1\ ]\!]_\Gamma = (?, ?)$. Suppose also that $[\![\ P_2\ ]\!]_\Gamma = (a_2, d_2)$ where $a_2 \in \{\mathsf{F}, ?\}$ (i.e $[\![\ P_2\ ]\!]_\Gamma$ is not a Permit). It follows that:

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_1 [\!]\!] \mathsf{P}_2\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_1 [\!]\!] \mathsf{P}_2\ ]\!]_\Gamma |\!\} \\
 & (? \vee a_2, (? - a_2) \vee (d_2 - ?)) \\
= & \{\!|\ \text{Definition of } a - b|\!\} \\
 & (? \vee a_2, (? \wedge \neg a_2) \vee (d_2 \wedge \neg ?)) \\
= & \{\!|\ \text{From the truth tables since } a_2 \in \{\mathsf{F}, ?\},\ \text{we have } (? \vee a_2) = ?|\!\} \\
 & (?, (? \wedge \neg a_2) \vee (d_2 \wedge \neg ?)) \\
= & \{\!|a_2 \in \{\mathsf{F}, ?\} \text{ then, from truth table, } \neg a_2 \in \{\mathsf{T}, ?\} \text{ and } (? \wedge \neg a) = ?|\!\} \\
 & (?, (? \vee (d_2 \wedge \neg ?)) \\
= & \{\!|\ \text{From the truth table } \neg ? = ?|\!\} \\
 & (?, (? \vee (d_2 \wedge ?)) \\
= & \{\!|\ \text{From the truth table } (d_2 \wedge ?) \in \{\mathsf{F}, ?\}|\!\} \\
 & (?, (? \vee (d_2 \wedge ?)) \\
= & \{\!|\ \text{From the truth tables since } (d_2 \wedge ?) \in \{\mathsf{F}, ?\} \text{ then } ? \vee (d_2 \wedge ?) = ?|\!\} \\
 & (?, ?) \\
= & \mathsf{Indeterminate(PD)}
\end{array}
$$

3. Let $[\![\ P_1\ ]\!]_\Gamma = (?, \mathsf{F}) = \mathsf{Indeterminate(P)}$ and $[\![\ P_2\ ]\!]_\Gamma = (\mathsf{F}, ?) = \mathsf{Indeterminate(D)}$ or $[\![\ P_2\ ]\!]_\Gamma = \mathsf{Deny} \in \{(\mathsf{F}, \mathsf{T}), (?, \mathsf{T})\}$

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_1 \| \mathsf{P}_2\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_1 \| \mathsf{P}_2\ ]\!]_\Gamma\ |\!\} \\
 & (? \vee a_2, (\mathsf{F} - a_2) \vee (d_2 - ?)) \\
= & \{\!|\ \text{Definition of } a - b\ |\!\} \\
 & (? \vee a_2, (\mathsf{F} \wedge \neg a_2) \vee (d_2 \wedge \neg ?)) \\
= & \{\!|\ \text{From the truth tables}\ |\!\} \\
 & (?, \mathsf{F} \vee (d_2 \wedge ?)) \\
= & \{\!|\ \text{From the truth tables}\ |\!\} \\
 & (?, (d_2 \wedge ?)) \\
= & \{\!|\ \text{From the truth tables since } d_2 \in \{\mathsf{T}, ?\}\ |\!\} \\
 & (?, ?) \\
= & \mathsf{Indeterminate(PD)}
\end{array}
$$

4. Suppose that $[\![\ P_1\ ]\!]_\Gamma = \mathsf{Indeterminate(P)}$ (The result is the same when we consider $[\![\ P_2\ ]\!]_\Gamma = \mathsf{Indeterminate(P)}$ since the operator $\|$ is commutative). We suppose that $[\![\ P_2\ ]\!]_\Gamma \neq \mathsf{Permit}$, $[\![\ P_2\ ]\!]_\Gamma \neq \mathsf{Indeterminate(D)}$, $[\![\ P_2\ ]\!]_\Gamma \neq \mathsf{Indeterminate(PD)}$ and $[\![\ P_2\ ]\!]_\Gamma \neq \mathsf{Deny}$. It means that $[\![\ P_1\ ]\!]_\Gamma = (?, \mathsf{F})$ and $[\![\ P_2\ ]\!]_\Gamma = \mathsf{N/A}$ or $[\![\ P_2\ ]\!]_\Gamma = \mathsf{Indeterminate(P)}$. It follows that $[\![\ P_1\ ]\!]_\Gamma = (?, \mathsf{F})$ and $[\![\ P_2\ ]\!]_\Gamma = (a_2, \mathsf{F})$

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_1 \| \mathsf{P}_2\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_1 \| \mathsf{P}_2\ ]\!]_\Gamma\ |\!\} \\
 & (? \vee a_2, (\mathsf{F} - a_2) \vee \mathsf{F} - ?)) \\
= & \{\!|\ \text{Definition of } a - b\ |\!\} \\
 & (? \vee a_2, (\mathsf{F} \wedge \neg a_2) \vee (\mathsf{F} \wedge \neg ?)) \\
= & \{\!|\ \text{From the truth tables}\ |\!\} \\
 & (?, \mathsf{F}) \\
= & \mathsf{Indeterminate(P)}
\end{array}
$$

5. Suppose that $[\![\ P_1\ ]\!]_\Gamma = \mathsf{Deny}$ (The result is the same when we consider $[\![\ P_2\ ]\!]_\Gamma = \mathsf{Deny}$ since the operator $\|$ is commutative). We suppose that $[\![\ P_2\ ]\!]_\Gamma \neq \mathsf{Permit}$, $[\![\ P_2\ ]\!]_\Gamma \neq \mathsf{Indeterminate(P)}$ and $[\![\ P_2\ ]\!]_\Gamma \neq \mathsf{Indeterminate(PD)}$. It means that $[\![\ P_1\ ]\!]_\Gamma = (\mathsf{F}, \mathsf{T})$ and $[\![\ P_2\ ]\!]_\Gamma = \mathsf{N/A}$, $[\![\ P_2\ ]\!]_\Gamma = \mathsf{Indeterminate(D)}$ or $[\![\ P_2\ ]\!]_\Gamma = \mathsf{Deny}$. It follows that $[\![\ P_1\ ]\!]_\Gamma = (\mathsf{F}, \mathsf{T})$ and $[\![\ P_2\ ]\!]_\Gamma = (\mathsf{F}, d_2)$

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_1 \llbracket \mathsf{P}_2\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_1 \llbracket \mathsf{P}_2\ ]\!]_\Gamma |\!\} \\
 & (\mathsf{F} \vee \mathsf{F}, (d_2 - \mathsf{F}) \vee \mathsf{T} - \mathsf{F})) \\
= & \{\!|\ \text{Definition of } a - b |\!\} \\
 & (\mathsf{F} \vee \mathsf{F}, (d_2 \wedge \neg\mathsf{F}) \vee (\mathsf{T} \wedge \neg\mathsf{F})) \\
= & \{\!|\ \text{From the truth tables } |\!\} \\
 & (\mathsf{F}, d_2 \vee \mathsf{T}) \\
= & \{\!|\ \text{From the truth tables } |\!\} \\
 & (\mathsf{F}, \mathsf{T}) \\
= & \mathsf{Deny}
\end{array}
$$

6. Suppose that $[\![\ P_1\ ]\!]_\Gamma = \mathsf{Indeterminate(D)}$ (The result is the same when we consider $[\![\ P_2\ ]\!]_\Gamma = \mathsf{Indeterminate(D)}$ since the operator $\llbracket$ is commutative). We suppose that $[\![\ P_2\ ]\!]_\Gamma \neq \mathsf{Permit}$, $[\![\ P_2\ ]\!]_\Gamma \neq \mathsf{Indeterminate(P)}$, $[\![\ P_2\ ]\!]_\Gamma \neq \mathsf{Indeterminate(PD)}$ and $[\![\ P_2\ ]\!]_\Gamma \neq \mathsf{Deny}$. It means that $[\![\ P_1\ ]\!]_\Gamma = (\mathsf{F}, ?)$ and $[\![\ P_2\ ]\!]_\Gamma = \mathsf{N/A}$ or $[\![\ P_2\ ]\!]_\Gamma = \mathsf{Indeterminate(D)}$. It follows that $[\![\ P_1\ ]\!]_\Gamma = (\mathsf{F}, ?)$ and $[\![\ P_2\ ]\!]_\Gamma = (\mathsf{F}, d_2)$ and $d_2 \in \{\mathsf{F}, ?\}$

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_1 \llbracket \mathsf{P}_2\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_1 \llbracket \mathsf{P}_2\ ]\!]_\Gamma |\!\} \\
 & (\mathsf{F} \vee \mathsf{F}, (d_2 - \mathsf{F}) \vee (? - \mathsf{F})) \\
= & \{\!|\ \text{Definition of } a - b |\!\} \\
 & (\mathsf{F} \vee \mathsf{F}, (d_2 \wedge \neg\mathsf{F}) \vee (? \wedge \neg\mathsf{F})) \\
= & \{\!|\ \text{From the truth tables } |\!\} \\
 & (\mathsf{F}, d_2 \vee ?) \\
= & \{\!|\ \text{From the truth tables since } d_2 \in \{\mathsf{F}, ?\} |\!\} \\
 & (\mathsf{F}, ?) \\
= & \mathsf{Indeterminate(D)}
\end{array}
$$

7. Otherwise $[\![\ P_1\ ]\!]_\Gamma = \mathsf{N/A} = (\mathsf{F}, \mathsf{F})$ and $[\![\ P_2\ ]\!]_\Gamma = \mathsf{N/A} = (\mathsf{F}, \mathsf{F})$.

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_1 \llbracket \mathsf{P}_2\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_1 \llbracket \mathsf{P}_2\ ]\!]_\Gamma |\!\} \\
 & (\mathsf{F} \vee \mathsf{F}, (\mathsf{F} - \mathsf{F}) \vee (\mathsf{F} - \mathsf{F})) \\
= & \{\!|\ \text{Definition of } a - b |\!\} \\
 & (\mathsf{F} \vee \mathsf{F}, (\mathsf{F} \wedge \neg\mathsf{F}) \vee (\mathsf{F} \wedge \neg\mathsf{F})) \\
= & \{\!|\ \text{From the truth tables } |\!\} \\
 & (\mathsf{F}, \mathsf{F}) \\
= & \mathsf{N/A}
\end{array}
$$

7.2. Deny-overrides:

Deny-overrides between $P_1, \ldots, P_n$, denoted by $DOR(P_1, \ldots, P_n)$: It denies if at least one policy denies and accepts if no one denies and at least one accepts. It can be formalized in SePL as

follows:

$$DOR(P_1, \ldots, P_n) \approx P_1 \| \ldots \| P_n$$

.

According to the XACML specification [27], we have:
The deny overrides combining algorithm is intended for those cases where a deny decision should have priority over a permit decision. This algorithm has the following behavior.
1. If any decision is "Deny", the result is "Deny".
2. Otherwise, if any decision is "IndeterminateDP", the result is "IndeterminateDP".
3. Otherwise, if any decision is "IndeterminateD" and another decision is ?IndeterminateP or Permit, the result is "IndeterminateDP".
4. Otherwise, if any decision is "IndeterminateD", the result is "IndeterminateD".
5. Otherwise, if any decision is "Permit", the result is "Permit".
6. Otherwise, if any decision is "IndeterminateP", the result is "IndeterminateP".
7. Otherwise, the result is "NotApplicable".

Our formalization feet with the XACML requirement and the proof is similar to the previous one.

7.3. First-Applicable:

First-Applicable between $P_1, \ldots, P_n$, denoted by $FA(P_1, \ldots, P_n)$: It accepts if there is at least one policy that accepts and that is not proceeded by a denying one and vise-versa. It can be formalized in SePL as follows:

$$FA(P_1, \ldots, P_n) \approx P_1 \ldots . P_n$$

.

According to the XACML specification [27], we have:
The following is a non-normative informative description of the "First-Applicable? rule-combining algorithm of a policy.
Each rule SHALL be evaluated in the order in which it is listed in the policy. For a particular rule, if the target matches and the condition evaluates to "True", then the evaluation of the policy SHALL halt and the corresponding effect of the rule SHALL be the result of the evaluation of the policy (i.e. "Permit" or "Deny"). For a particular rule selected in the evaluation, if the target evaluates to "False" or the condition evaluates to "False", then the next rule in the order SHALL be evaluated. If no further rule in the order exists, then the policy SHALL evaluate to "NotApplicable".
If an error occurs while evaluating the target or condition of a rule, then the evaluation SHALL halt, and the policy shall evaluate to "Indeterminate", with the appropriate error status.

We consider the following cases:

1. First one accept: Suppose that $[\![P_1]\!]_\Gamma = \mathsf{Permit} = (T, d_1)$ and $[\![P_2]\!]_\Gamma = (a_2, d_2)$ where $d_1 \in \{F, ?\}$. It follows that:

$$\begin{array}{ll} & [\![\mathsf{P_1} \cdot \mathsf{P_2}]\!]_\Gamma) \\ = & \{|\text{Definition of } [\![\mathsf{P_1} \cdot \mathsf{P_2}]\!]_\Gamma|\} \\ & (T \vee (a_2 - d_1), d_1 \vee (d_2 - T)) \\ = & \{|\text{From the truth tables }|\} \\ & (T, d_1 \vee F) \\ = & \{|\text{From the truth tables }|\} \\ & (T, d_1) \\ = & \mathsf{Permit} \end{array}$$

2. First one deny: Suppose that $[\![P_1]\!]_\Gamma = \mathsf{Deny} = (a_1, T)$ and $[\![P_2]\!]_\Gamma = (a_2, d_2)$ where $a_1 \in \{F, ?\}$. It follows that:

$$\begin{array}{ll} & [\![\mathsf{P_1} \cdot \mathsf{P_2}]\!]_\Gamma) \\ = & \{|\text{Definition of } [\![\mathsf{P_1} \cdot \mathsf{P_2}]\!]_\Gamma|\} \\ & (a_1 \vee (a_2 - T), T \vee (d_2 - a_1)) \\ = & \{|\text{From the truth tables }|\} \\ & (a_1 \vee F, T) \\ = & \{|\text{From the truth tables }|\} \\ & (a_1, T) \\ = & \mathsf{Deny} \end{array}$$

3. First one not applicable: Suppose that $[\![P_1]\!]_\Gamma = \mathsf{N/A} = (F, F)$ and $[\![P_2]\!]_\Gamma = (a_2, d_2)$. It follows that:

$$\begin{array}{ll} & [\![\mathsf{P_1} \cdot \mathsf{P_2}]\!]_\Gamma) \\ = & \{|\text{Definition of } [\![\mathsf{P_1} \cdot \mathsf{P_2}]\!]_\Gamma|\} \\ & (F \vee (a_2 - F), F \vee (d_2 - F)) \\ = & \{|\text{From the truth tables }|\} \\ & (F \vee a_2, F \vee d_2) \\ = & \{|\text{From the truth tables }|\} \\ & (a_2, d_2) \\ = & [\![P_2]\!]_\Gamma \end{array}$$

4. First one indeterminate(P): Suppose that $[\![P_1]\!]_\Gamma = \mathsf{Indeterminate(P)} = (?, F)$ and $[\![P_2]\!]_\Gamma = (a_2, d_2)$. It follows that:

$$
\begin{array}{ll}
 & [\![\ \mathsf{P_1} \cdot \mathsf{P_2}\ ]\!]_\Gamma) \\
= & \{|\text{Definition of } [\![\ \mathsf{P_1} \cdot \mathsf{P_2}\ ]\!]_\Gamma|\} \\
 & (? \vee (a_2 - F), F \vee (d_2 - ?) \\
= & \{|\text{From the truth tables }|\} \\
 & (? \vee a_2, d_2 \wedge ?) \\
= & \{|\text{From the truth tables }|\} \\
 & (?, ?) \\
= & \mathsf{Indeterminate(PD)}
\end{array}
$$

5. First one indeterminate(D): Suppose that $[\![\ P_1\ ]\!]_\Gamma = \mathsf{Indeterminate(D)} = (F, ?)$ and $[\![\ P_2\ ]\!]_\Gamma = (a_2, d_2)$. It follows that:

$$
\begin{array}{ll}
 & [\![\ \mathsf{P_1} \cdot \mathsf{P_2}\ ]\!]_\Gamma) \\
= & \{|\text{Definition of } [\![\ \mathsf{P_1} \cdot \mathsf{P_2}\ ]\!]_\Gamma|\} \\
 & (F \vee (a_2 - ?), ? \vee (d_2 - F) \\
= & \{|\text{From the truth tables }|\} \\
 & (a_2 \wedge ?, ? \vee d_2) \\
= & \{|\text{From the truth tables }|\} \\
 & (?, ?) \\
= & \mathsf{Indeterminate(PD)}
\end{array}
$$

6. First one indeterminate(PD): Suppose that $[\![\ P_1\ ]\!]_\Gamma = \mathsf{Indeterminate(PD)} = (?, ?)$ and $[\![\ P_2\ ]\!]_\Gamma = (a_2, d_2)$. It follows that:

$$
\begin{array}{ll}
 & [\![\ \mathsf{P_1} \cdot \mathsf{P_2}\ ]\!]_\Gamma) \\
= & \{|\text{Definition of } [\![\ \mathsf{P_1} \cdot \mathsf{P_2}\ ]\!]_\Gamma|\} \\
 & (? \vee (a_2 - ?), ? \vee (d_2 - ?) \\
= & \{|\text{From the truth tables }|\} \\
 & (? \vee (a_2 \wedge ?), ? \vee (? \wedge d_2)) \\
= & \{|\text{From the truth tables }|\} \\
 & (? \vee ?, ? \vee ?) \\
= & \{|\text{From the truth tables }|\} \\
 & (?, ?) \\
= & \mathsf{Indeterminate(PD)}
\end{array}
$$

7.4. Only-one-applicable:

Only-one-applicable between $P_1, \ldots, P_n$, denoted by $OOA(P_1, \ldots, P_n)$: if more than one policy is applicable, the result will be neither accept nor deny (without decision). Otherwise, the unique applicable policy will be applied.

It can be formalized in SePL as follows:

$$OOA(P_1, \ldots, P_n) \approx (P_1 \ominus \Sigma_{i=2}^{n} P_i) + \ldots + (P_j \ominus \Sigma_{i=1, i \neq j}^{n} P_i) + \ldots + (P_n \ominus \Sigma_{i=1}^{n-1} P_i)$$

.

According to the XACML specification [27], we have:

" In the entire set of policies in the policy set, if no policy is considered applicable by virtue of its target, then the result of the policy-combination algorithm SHALL be "NotApplicable". If more than one policy is considered applicable by virtue of its target, then the result of the policy combination algorithm SHALL be "Indeterminate".
If only one policy is considered applicable by evaluation of its target, then the result of the policy-combining algorithm SHALL be the result of evaluating the policy.
If an error occurs while evaluating the target of a policy, or a reference to a policy is considered invalid or the policy evaluation results in "Indeterminate, then the policy set SHALL evaluate to "Indeterminate", with the appropriate error status."

We consider the following cases:

1. No one applicable: Suppose that $[\![P_1]\!]_\Gamma = \mathsf{N/A} = (F, F)$ and $[\![P_2]\!]_\Gamma = \mathsf{N/A} = (F, F)$. It follows that:

$$\begin{array}{ll} & [\![\mathsf{P}_1 \ominus \mathsf{P}_2]\!]_\Gamma \\ = & \{\!| \text{ Definition of } [\![\mathsf{P}_1 \ominus \mathsf{P}_2]\!]_\Gamma |\!\} \\ & (F \ominus (F \vee F), F \ominus (F \vee F)) \\ = & \{\!|\text{From the truth tables }|\!\} \\ & (F \ominus F, F \ominus F) \\ = & \{\!|\text{From the truth tables }|\!\} \\ & (F, F) \\ \Rightarrow & [\![(\mathsf{P}_1 \ominus \mathsf{P}_2) + (\mathsf{P}_2 \ominus \mathsf{P}_1)]\!]_\Gamma \\ & ((F, F) + (F, F)) \\ = & \{\!|\text{From the truth tables }|\!\} \\ & (F \vee F, F \vee F) \\ = & \{\!|\text{From the truth tables }|\!\} \\ & (F, F) \\ = & \mathsf{N/A} \end{array}$$

2. More than one applicable: We distinguish the following cases:

   - $[\![P_1]\!]_\Gamma = \mathsf{Permit} = (T, d_1)$ and $[\![P_2]\!]_\Gamma = \mathsf{Permit} = (T, d_2)$ where $\{d_1, d_2\} \subseteq \{F, ?\}$. It follows that:

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma |\!\} \\
 & (T \ominus (T \vee d_2), d_1 \ominus (T \vee d_2)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (T \ominus T, d_1 \ominus T)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (?, d_1) \\
\Rightarrow & [\![\ (\mathsf{P}_1 \ominus \mathsf{P}_2) + (\mathsf{P}_2 \ominus \mathsf{P}_1)\ ]\!]_\Gamma \\
 & (?, d_1) + (?, d_2) \\
= & \{\!|\text{From the definition of + }|\!\} \\
 & (? \vee d_1, ? \vee d_2) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (?, ?) \\
= & \mathsf{Indeterminate(PD)}
\end{array}
$$

- $[\![\ P_1\ ]\!]_\Gamma = \mathsf{Deny} = (d_1, T)$ and $[\![\ P_2\ ]\!]_\Gamma = \mathsf{Permit} = (T, d_2)$ where $\{d_1, d_2\} \subseteq \{F, ?\}$ (The result will be the same for the symmetric case). It follows that:

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma |\!\} \\
 & (d_1 \ominus (T \vee d_2), T \ominus (T \vee d_2)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (d_1 \ominus T, T \ominus T)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (d_1, ?)
\end{array}
$$

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_2 \ominus \mathsf{P}_1\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma |\!\} \\
 & (T \ominus (T \vee d_1), d_2 \ominus (T \vee d_1)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (T \ominus T, d_2 \ominus T)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (?, d_2)
\end{array}
$$

$$
\begin{array}{ll}
\Rightarrow & [\![\ (\mathsf{P}_1 \ominus \mathsf{P}_2) + (\mathsf{P}_2 \ominus \mathsf{P}_1)\ ]\!]_\Gamma \\
 & (d_1, ?) + (?, d_2) \\
= & \{\!|\text{From the definition of + }|\!\} \\
 & (d_1 \vee ?, ? \vee d_2) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (?, ?) \\
= & \mathsf{Indeterminate(PD)}
\end{array}
$$

- $[\![P_1]\!]_\Gamma = \mathsf{Deny} = (d_1, T)$ and $[\![P_2]\!]_\Gamma = \mathsf{Deny} = (d_2, T)$ where $\{d_1, d_2\} \subseteq \{F, ?\}$. It follows that:

$$
\begin{array}{ll}
 & [\![\mathsf{P}_1 \ominus \mathsf{P}_2]\!]_\Gamma \\
= & \{\!| \text{ Definition of } [\![\mathsf{P}_1 \ominus \mathsf{P}_2]\!]_\Gamma |\!\} \\
 & (d_1 \ominus (T \vee d_2), T \ominus (T \vee d_2)) \\
= & \{\!| \text{From the truth tables } |\!\} \\
 & (d_1 \ominus T, T \ominus T)) \\
= & \{\!| \text{From the truth tables } |\!\} \\
 & (d_1, ?) \\
\Rightarrow & [\![(\mathsf{P}_1 \ominus \mathsf{P}_2) + (\mathsf{P}_2 \ominus \mathsf{P}_1)]\!]_\Gamma \\
 & (d_1, ?) + (d_2, ?) \\
= & \{\!| \text{From the definition of + } |\!\} \\
 & (d_1 \vee d_2, ? \vee ?) \\
= & \{\!| \text{From the truth tables } |\!\} \\
 & (d_1 \vee d_2, ?) \\
\in & \{\mathsf{Indeterminate(PD)}, \mathsf{Indeterminate(D)}\}
\end{array}
$$

- $[\![P_1]\!]_\Gamma = \mathsf{Deny} = (d_1, T)$ and $[\![P_2]\!]_\Gamma = \mathsf{N/A} = (F, F)$ where $d_1 \in \{F, ?\}$ (The result will be the same for the symmetric case). It follows that:

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma |\!\} \\
 & (d_1 \ominus (F \vee F), T \ominus (F \vee F)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (d_1 \ominus F, T \ominus F)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (d_1, T)
\end{array}
$$

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_2 \ominus \mathsf{P}_1\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_2 \ominus \mathsf{P}_1\ ]\!]_\Gamma |\!\} \\
 & (F \ominus (T \vee d_1), F \ominus (F \vee d_1)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (F \ominus T, F \ominus T)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (F, F)
\end{array}
$$

$$
\begin{array}{ll}
\Rightarrow & [\![\ (\mathsf{P}_1 \ominus \mathsf{P}_2) + (\mathsf{P}_2 \ominus \mathsf{P}_1)\ ]\!]_\Gamma \\
 & (d_1, T) + (F, F) \\
= & \{\!|\text{From the definition of + }|\!\} \\
 & (d_1 \vee F, T \vee F) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (d_1, T) \\
= & \mathsf{Deny}
\end{array}
$$

- $[\![\ P_1\ ]\!]_\Gamma = \mathsf{Deny} = (d_1, T)$ and $[\![\ P_2\ ]\!]_\Gamma = \mathsf{Indeterminate} = (?, d_2)$ where $\{d_1, d_2\} \subseteq \{F, ?\}$ (The result will be the same for the symmetric case). It follows that:

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma |\!\} \\
 & (d_1 \ominus (d_2 \vee ?), T \ominus (d_2 \vee ?)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (d_1 \ominus ?, T \ominus ?)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (d_1, ?)
\end{array}
$$

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_2 \ominus \mathsf{P}_1\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_2 \ominus \mathsf{P}_1\ ]\!]_\Gamma |\!\} \\
 & (? \ominus (T \vee d_1), d_2 \ominus (T \vee d_1)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (? \ominus T, d_2 \ominus T) \\
= & \{\!|\text{From the truth tables since } d_2 \in \{F, ?\}|\!\} \\
 & (?, d_2)
\end{array}
$$

$$
\begin{array}{ll}
\Rightarrow & [\![\ (\mathsf{P}_1 \ominus \mathsf{P}_2) + (\mathsf{P}_2 \ominus \mathsf{P}_1)\ ]\!]_\Gamma \\
 & (d_1, ?) + (?, d_2) \\
= & \{\!|\text{From the definition of + }|\!\} \\
 & (d_1 \vee ?, ? \vee d_2) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (?, ?) \\
= & \mathsf{Indeterminate}(\mathsf{PD})
\end{array}
$$

- $[\![\ P_1\ ]\!]_\Gamma = \mathsf{Deny} = (d_1, T)$ and $[\![\ P_2\ ]\!]_\Gamma = \mathsf{Indeterminate}(\mathsf{D}) = (F, ?)$ where $d_1 \in \{F, ?\}$ (The result will be the same for the symmetric case). It follows that:

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma \\
= & \{|\ \text{Definition of } [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma |\} \\
 & (d_1 \ominus (F \vee ?), T \ominus (F \vee ?)) \\
= & \{|\text{From the truth tables }|\} \\
 & (d_1 \ominus ?, T \ominus ?)) \\
= & \{|\text{From the truth tables }|\} \\
 & (d_1, ?)
\end{array}
$$

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_2 \ominus \mathsf{P}_1\ ]\!]_\Gamma \\
= & \{|\ \text{Definition of } [\![\ \mathsf{P}_2 \ominus \mathsf{P}_1\ ]\!]_\Gamma |\} \\
 & (F \ominus (T \vee d_1), ? \ominus (T \vee d_1)) \\
= & \{|\text{From the truth tables }|\} \\
 & (F \ominus T, ? \ominus T) \\
= & \{|\text{From the truth tables }|\} \\
 & (F, ?)
\end{array}
$$

$$
\begin{array}{ll}
\Rightarrow & [\![\ (\mathsf{P}_1 \ominus \mathsf{P}_2) + (\mathsf{P}_2 \ominus \mathsf{P}_1)\ ]\!]_\Gamma \\
 & (d_1, ?) + (F, ?) \\
= & \{|\text{From the definition of + }|\} \\
 & (d_1 \vee F, ? \vee ?) \\
= & \{|\text{From the truth tables }|\} \\
 & (d_1, ?) \\
\in & \{\mathsf{Indeterminate(PD)}, \mathsf{Indeterminate(D)}\}
\end{array}
$$

- $[\![\ P_1\ ]\!]_\Gamma = \mathsf{Permit} = (T, d_1)$ and $[\![\ P_2\ ]\!]_\Gamma = \mathsf{Indeterminate} = (?, d_2)$ where $\{d_1, d_2\} \subseteq \{F, ?\}$ (The result will be the same for the symmetric case). It follows that:

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma |\!\} \\
 & (T \ominus (d_2 \vee ?), d_1 \ominus (d_2 \vee ?)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (T \ominus ?, d_1 \ominus ?) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (?, d_1)
\end{array}
$$

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_2 \ominus \mathsf{P}_1\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_2 \ominus \mathsf{P}_1\ ]\!]_\Gamma |\!\} \\
 & (? \ominus (T \vee d_1), d_2 \ominus (T \vee d_1)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (? \ominus T, d_2 \ominus T)) \\
= & \{\!|\text{From the truth tables since } d_2 \in \{F, ?\}|\!\} \\
 & (?, d_2)
\end{array}
$$

$$
\begin{array}{ll}
\Rightarrow & [\![\ (\mathsf{P}_1 \ominus \mathsf{P}_2) + (\mathsf{P}_2 \ominus \mathsf{P}_1)\ ]\!]_\Gamma \\
 & (?, d_1) + (?, d_2) \\
= & \{\!|\text{From the definition of + }|\!\} \\
 & (? \vee ?, d_1 \vee d_2) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (?, d_1 \vee d_2) \\
\in & \{\mathsf{Indeterminate(PD)}, \mathsf{Indeterminate(D)}\}
\end{array}
$$

- $[\![\ P_1\ ]\!]_\Gamma = \mathsf{Permit} = (T, d_1)$ and $[\![\ P_2\ ]\!]_\Gamma = \mathsf{Indeterminate(D)} = (F, ?)$ where $d_1 \in \{F, ?\}$ (The result will be the same for the symmetric case). It follows that:

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_1 \ominus \mathsf{P}_2\ ]\!]_\Gamma |\!\} \\
 & (T \ominus (F \vee ?), d_1 \ominus (F \vee ?)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (T \ominus ?, d_1 \ominus ?) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (?, d_1)
\end{array}
$$

$$
\begin{array}{ll}
 & [\![\ \mathsf{P}_2 \ominus \mathsf{P}_1\ ]\!]_\Gamma \\
= & \{\!|\ \text{Definition of } [\![\ \mathsf{P}_2 \ominus \mathsf{P}_1\ ]\!]_\Gamma |\!\} \\
 & (F \ominus (T \vee d_1), ? \ominus (T \vee d_1)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (F \ominus T, ? \ominus T)) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (F, ?)
\end{array}
$$

$$
\begin{array}{ll}
\Rightarrow & [\![\ (\mathsf{P}_1 \ominus \mathsf{P}_2) + (\mathsf{P}_2 \ominus \mathsf{P}_1)\ ]\!]_\Gamma \\
 & (?, d_1) + (F, ?) \\
= & \{\!|\text{From the definition of + }|\!\} \\
 & (? \vee F, ? \vee d_1) \\
= & \{\!|\text{From the truth tables }|\!\} \\
 & (?, ?) \\
= & \mathsf{Indeterminate}(\mathsf{PD})
\end{array}
$$

7.5. Deny-unless-permit

Deny-unless-permit between $P_1, \ldots, P_n$ is formalized in SePL as follows: denoted

$$DUP(P_1, \ldots, P_n) \approx \ulcorner(P_1 \llbracket \ldots \llbracket P_n)$$

.

According to the XACML specification [27], we have:

The "Deny-unless-permit" combining algorithm is intended for those cases where a permit decision should have priority over a deny decision, and an "Indeterminate" or "NotApplicable" must never be the result. It is particularly useful at the top level in a policy structure to ensure that a PDP will always return a definite "Permit" or "Deny" result. This algorithm has the following behavior.

1. If any decision is "Permit", the result is "Permit".

2. Otherwise, the result is "Deny".

1. Suppose that $P = P_1 \llbracket \ldots \llbracket P_n$ and it exists i, $i \in \{1, \ldots, n\}$, such that $[\![\ P_i\ ]\!]_\Gamma = \mathsf{Permit}$.

From the definition of **Permit-override**, wee have that $[\![P]\!]_\Gamma = \mathsf{Permit} = (\mathsf{T}, d_1)$ where $d_1 \in \{\mathsf{F}, ?\}$. It follows, from the definition of $\ulcorner$, that $[\![\ulcorner P]\!]_\Gamma == (\mathsf{T}, F)$

$$
\begin{array}{ll}
 & [\![\ulcorner \mathsf{P}.0]\!]_\Gamma \\
= & \{|\text{From the definition of } [\![P.Q]\!]|\} \\
 & (T \vee (F - T), F \vee (T - T))) \\
= & \{|\text{From the truth Table }|\} \\
 & (T, F \vee (T \wedge \neg T))) \\
= & \{|\text{From the truth Table }|\} \\
 & (T, F)) \\
= & \{|\text{Definition of } \mathsf{Permit}\ |\} \\
 & \mathsf{Permit}
\end{array}
$$

2. Suppose that $P = P_1 [\!] \ldots [\!] P_n$ and for all i, $i \in \{1, \ldots, n\}$, we have $[\![P_i]\!]_\Gamma \neq \mathsf{Permit}$. From the definition of **Permit-override**, we have $[\![P]\!]_\Gamma \in \{(?, ?), (?, F), (F, ?), (F, F), (F, T)\}$. It follows that for the definition of $\ulcorner$ that $[\![\ulcorner P]\!]_\Gamma \in \{(F, F), (F, T)\}$, i.e $[\![\ulcorner P]\!]_\Gamma = (F, d)$ where $d \in \{F, T\}$

$$
\begin{array}{ll}
 & [\![\ulcorner \mathsf{P}.0]\!]_\Gamma \\
= & \{|\text{From the definition of } [\![P.Q]\!]|\} \\
 & (F \vee (F - d), d \vee (T - F))) \\
= & \{|\text{From the truth Table }|\} \\
 & (F, d \vee T) \\
= & \{|\text{From the truth Table }|\} \\
 & (F, T)) \\
= & \{|\text{Definition of } \mathsf{Deny}\ |\} \\
 & \mathsf{Deny}
\end{array}
$$

7.6. Permit-unless-deny

Permit-unless-deny between $P_1, \ldots, P_n$ is formalized in SePL as follows:

$$DUP(P_1, \ldots, P_n) \approx \ulcorner (P_1 [\!] \ldots [\!] P_n).1$$

According to the XACML specification [27], we have:

The "Permit-unless-deny" combining algorithm is intended for those cases where a deny decision should have priority over a permit decision, and an "Indeterminate" or "NotApplicable" must never be the result. It is particularly useful at the top level in a policy structure to ensure that a PDP will always return a definite "Permit" or "Deny" result. This algorithm has the following behavior.

1. If any decision is "Deny", the result is "Deny".

2. Otherwise, the result is "Permit".

1. Suppose that $P = P_1]\!] \ldots]\!] P_n$ and it exists i, $i \in \{1, \ldots, n\}$, such that $[\![P_i]\!]_\Gamma = \mathsf{Deny}$. From the definition of **Deny-override**, wee have that $[\![P]\!]_\Gamma = \mathsf{Deny} = (d_1, \mathsf{T})$ where $d_1 \in \{\mathsf{F}, ?\}$. It follows, from the definition of $\ulcorner$ that $[\![\ulcorner P]\!]_\Gamma = (F, \mathsf{T})$

$$
\begin{array}{ll}
 & [\![\ulcorner \mathsf{P}.1]\!]_\Gamma \\
= & \{|\text{From the definition of } [\![P.Q]\!]|\} \\
 & (F \vee (T - T), T \vee (F - F))) \\
= & \{|\text{From the truth Table }|\} \\
 & (F, T) \\
= & \{|\text{Definition of } \mathsf{Deny} \,|\} \\
 & \mathsf{Deny}
\end{array}
$$

2. Suppose that $P = P_1]\!] \ldots]\!] P_n$ and for all i, $i \in \{1, \ldots, n\}$, we have $[\![P_i]\!]_\Gamma \neq \mathsf{Deny}$. From the definition of **Permit-override**, wee have that $[\![P]\!]_\Gamma \in \{(?, ?), (?, F), (F, ?), (F, F), (T, F)\}$. It follows that for the definition of $\ulcorner$ that $[\![\ulcorner P]\!]_\Gamma \in \{(F, F), (T, F)\}$, i.e $[\![\ulcorner P]\!]_\Gamma = (a, F)$ where $a \in \{F, T\}$

$$
\begin{array}{ll}
 & [\![\ulcorner \mathsf{P}.0]\!]_\Gamma \\
= & \{|\text{From the definition of } [\![P.Q]\!]|\} \\
 & (a \vee (T - F), F \vee (F - d))) \\
= & \{|\text{From the truth Table }|\} \\
 & (a \vee T, F) \\
= & \{|\text{From the truth Table }|\} \\
 & (T, F)) \\
= & \{|\text{Definition of } \mathsf{Permit} \,|\} \\
 & \mathsf{Permit}
\end{array}
$$